Introduction au Webmapping
TD Leaflet côté client
et PostgreSQL / PostGIS côté serveur

Windows 10 Windows 11 macOS

Apache Web Server

HTML CSS JAVASCRIPT

Leaflet

php PostgreSQL PostGIS

TD 1er année ingénieur

MODULE D'OUVERTURE : WEBMAPPING 2022-2023

Jean-Marc Gilliot

jean-marc.gilliot@agroparistech.fr

AgroParisTech

Grande école européenne d'ingénieurs et de managers
dans le domaine du vivant et de l'environnement

 Conseil, astuce, info  A vous de jouer  Attention

Table des matières

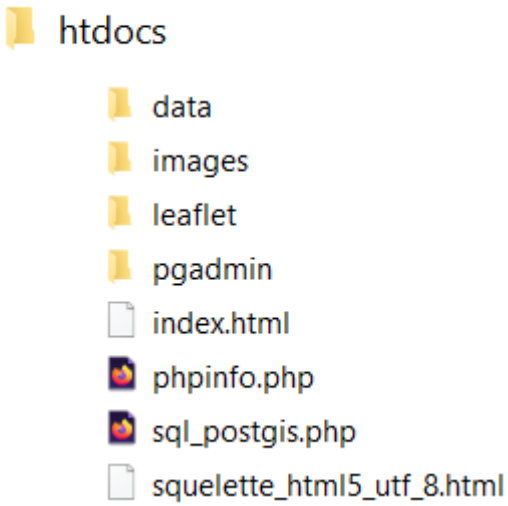
Mise en place des données sur le serveur

Voir PDF de l'installation du TD

les deux sous-dossiers : leaflet et data doivent être dans htdocs

Les dossier leaflet et data sont donc à la racine de votre site.

Le dossier htdocs est la racine du site WEB pour le serveur apache



Le dossier BD_IDF est dans vos Documents (pas dans l'espace web)

Consultez : https://leafletjs.com/

https://www.postgresql.org/

https://postgis.net/

II. Introduction à l'outil cartographique Leaflet en javascript

1. Première carte Leaflet dans une page html

● Créer une nouvelle page à la racine de votre site de nom : leaflet.html avec les sections classiques :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
leaflet
  </body>
</html>
```

Dans la section <head> de la page html, remarquer l'encodage des caractères à UTF-8, pour avoir les caractères accentués :

● Dans la section <head> de la page html, charger la librairie javascript leaflet et charger le style « CSS » de Leaflet :

```
<script src="/leaflet/leaflet.js"></script>

<link rel="stylesheet" href="/leaflet/leaflet.css" type="text/css">
```

 Il est aussi possible de charger leaflet depuis un serveur sur le net, sans avoir besoin d'en avoir une copie sur son propre serveur, avec le type de lien ci-dessous :

```
<link rel="stylesheet" href="https://unpkg.com/leaflet@1.4.0/dist/leaflet.css" />
```

● Création du conteneur de la carte = un tag dans la section <BODY>

```
<div id="map" style="width: 600px; height: 400px"></div>
```

C'est l'endroit où va être affichée la carte dans la page, id= 'map' est l'identifiant de la carte.

● Créer une section <script> dans <body> après le où l'on va insérer le code javascript de création de la carte, avec une première couche qui est la carte « OpenStreetMap » :

```
<script>
  // création de l'objet carte
  var map = L.map('map');
  // Centrer la vue sur Paris lat=48.853, long=2.348 et zoom=12
  map.setView([48.853,2.348],12);
  // création d'une couche OpenStreetMap (osm)
  var osmLayer = L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png');
  // Ajout de la couche osm à la carte
  osmLayer.addTo(map); // ou map.addLayer(osmLayer);
</script>
```

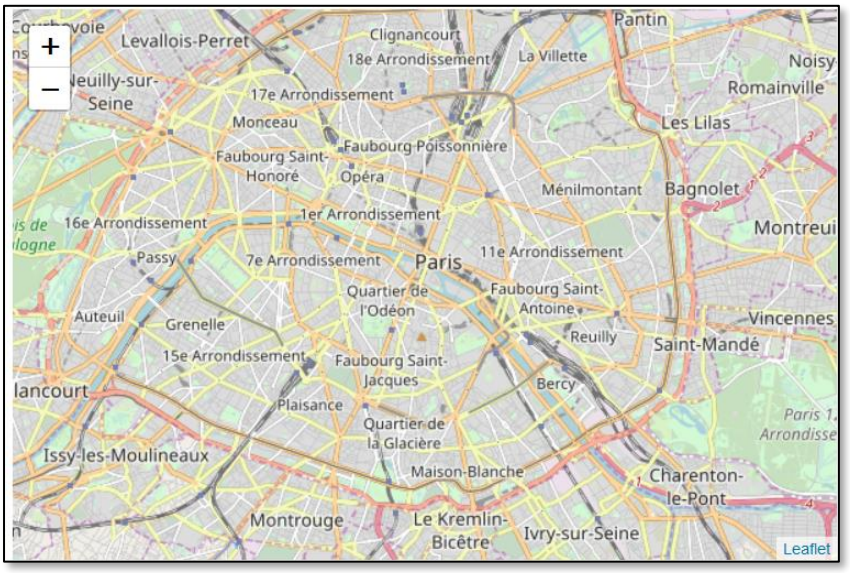
« L » fait référence à la librairie Leaftlet

L.map() appelle la fonction de création d'une carte (map) de Leaftlet

L.tileLayer() appelle une fonction de création de couche (layer)

```
map.removeLayer(osmLayer)    permet d'enlever une couche
```

Le résultat :



Zoomer  et scroller avec la main  dans votre carte

2. Ajouter des couches de base ou « Base Layers »

Leaflet distingue **deux types de couches** : les ***base layers*** et les ***Overlay layers***. Les Base layers sont des cartes générales de base, ou couches de fond, qui servent principalement à se repérer, telle que OpenSteetMap vu au point 1. Il y a généralement qu'une seule Base layer visible à un moment donné. Les Overlay layers sont les couches « métiers », les couches de l'application, on peur en superposer plusieurs, vecteurs et raster.

■ *Accès aux couches IGN du Géoportail*

Normalement il faut créer un compte géoservice pour accéder aux flux du Géoportail IGN avec un code (**clé**), depuis peu l'IGN met en place des accès simplifiés sans compte aux flux du Géoportail avec une clé générique.

Pour voir tous les accès possibles :

https://geoservices.ign.fr/ Consulter la section 

Services web

Il y a des exemples de code avec LeafLet dans ces pages IGN

Exemples des couches su service Web « découverte » 

La clé d'accès est alors soit : « decouverte » ou « pratique » et doit apparaitre dans le chemin d'accès :

https://wxs.ign.fr/decouverte/geoportail

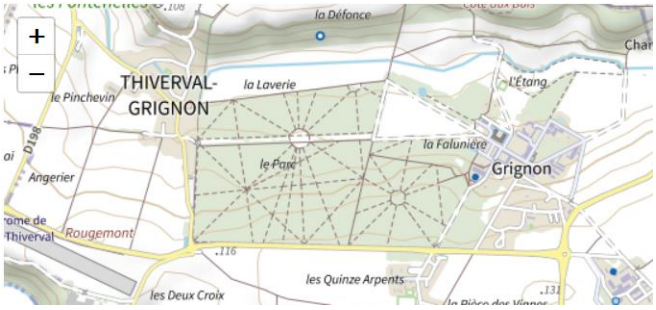
- Orthophotos IGN : LAYER=**ORTHOIMAGERY.ORTHOPHOTOS**

```
L.tileLayer('https://wxs.ign.fr/decouverte/geoportail/wmts?REQUEST=GetTile&SERVICE=WMTS
&VERSION=1.0.0&STYLE=normal&TILEMATRIXSET=PM&FORMAT=image/jpeg&LAYER=ORTHOIMAGERY.ORTHO
PHOTOS&TILEMATRIX={z}&TILEROW={y}&TILECOL={x}');
```

Ajouter une variable ignLayer avec cette couche

- IGN PLAN : cartes IGN : LAYER=**GEOGRAPHICALGRIDSYSTEMS.PLANIGNV2**

```
L.tileLayer('https://wxs.ign.fr/decouverte/geoportail/wmts?REQUEST=GetTile&SERVICE=WMTS
&VERSION=1.0.0&STYLE=normal&TILEMATRIXSET=PM&FORMAT=image/png&LAYER=GEOGRAPHICALGRIDSYS
TEMS.PLANIGNV2&TILEMATRIX={z}&TILEROW={y}&TILECOL={x}');
```



- IGN cadastre BD parcellaire : LAYER=**CADASTRALPARCELS.PARCELS**

Ici avec la clé « parcellaire »

```
L.tileLayer('https://wxs.ign.fr/parcellaire/geoportail/wmts?REQUEST=GetTile&SERVICE=WMT
S&VERSION=1.0.0&STYLE=bdparcellaire&TILEMATRIXSET=PM&FORMAT=image/png&LAYER=CADASTRALPA
RCELS.PARCELS&TILEMATRIX={z}&TILEROW={y}&TILECOL={x}');
```

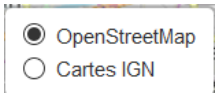
l'extension Leaflet de l'IGN : GpPluginLeaflet est aussi un moyen possible pour accéder aux fonds IGN (moins pratique).

(Il faut dans le code javascript appeler la fonction Gp.Services.getConfig() avec la clé pour accéder aux fonds IGN. En cas de succès (OnSuccess) une fonction est appellée, on doir encapsuler tout notre code javascript de Leaflet dans cette fonction.)

■ *Ajout d'un contrôle de couches pour plusieurs Base Layers*

```
var layerSwitcher = L.control.layers({"OpenStreetMap": osmLayer, "Cartes IGN": ignLayer
});

layerSwitcher.addTo(map);
```

Ce bouton  est ajouté en haut à droite de la carte.

En cliquant sur le bouton le menu des couches apparaît : 

on peut ainsi choisir d'afficher l'une ou l'autre des Base layers.

■ *Quelques ressources supplémentaires de Base layers :*

OpenStreetMap en noir et blanc

http://{s}.www.toolserver.org/tiles/bw-mapnik/{z}/{x}/{y}.png

OpenTopoMap

https://{s}.tile.opentopomap.org/{z}/{x}/{y}.png

ESRI World Imagery

https://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/MapServer/tile/{z}/{y}/{x}

■ *Ajout d'un contrôle d'affichage plein écran « fullscreen »*

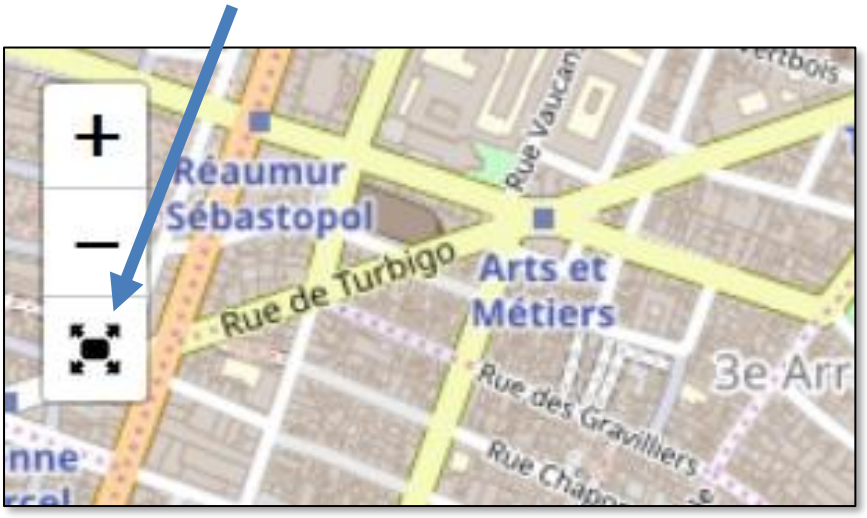
On a utilisé ici une extension leaflet de type « fullscreen » (il en existe plusieurs), on l'active dans la section <head> :

```
<script src="leaflet/fullscreen/Control.FullScreen.js"></script>
<link rel="stylesheet" href="leaflet/fullscreen/Control.FullScreen.css" />
```

On va créer un objet « contrôle » pour ajouter le bouton de plein écran : 

```
// Création du contrôle fullscreen
var fsControl = new L.Control.FullScreen();
// Ajout du bouton fullscreen sur la carte
map.addControl(fsControl);
```

Le bouton apparait en haut à gauche de la carte, sous les boutons de zoom :



On peut alors basculer entre le mode d'affichage normal et le mode plein écran, en cliquant sur ce bouton.

Pour enlever un Contrôle : **map.removeControl(fsControl)** ;

On verra par la suite d'autres types de contrôle.

3. Ajouter des couches « Overlays »

■ *Ajouter la couche Overlay vecteur des nouvelles régions en format geoJSON*

Dans le dossier data, regions.geojson est la couche des nouvelles régions françaises, en format geoJSON.

Attention les coordonnées en geoJSON sont en [longitude, latitude] WGS84

Dans le code javascript de la carte :

```
function readGeojson(fichier){
 var xmlhttp = new XMLHttpRequest();
 xmlhttp.open("GET",fichier , false);
 xmlhttp.overrideMimeType("application/json");
 xmlhttp.send();
 return JSON.parse(xmlhttp.responseText);
}

var geojsonLayer = new L.GeoJSON(readGeojson("data/regions.geojson"));
geojsonLayer.addTo(map);
```

La fonction readGeojson pour lire un fichier geoJson sur le serveur :
xmlhttp sert à envoyer une requête http au serveur pour télécharger le fichier geojson.
JSON.parse construite l'objet javascript à partir de la chaîne de caractères renvoyée.
L.geoJSON construit la couche en format geoJSON.

Le résultat :

JSON.stringify(objJSON) est la fonction inverse de JSON.parse, qui construit un texte à partir de l'objet JSON. On peut ainsi à l'inverse uploader sur le serveur une couche geoJSON dans un fichier, en appelant un script php sur le serveur, il faudra avoir les permissions en écriture sur le dossier du serveur.

```php
<?php
// exemple fichier php pour traiter l'upload : saveJSON.php
$str=$_POST['json'];
$file = fopen("vols.json","w");
fwrite($file,$str);
fclose($file);
?>
```

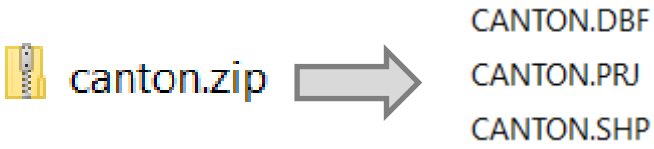
■ *Ajouter une couche Overlay vecteur en format shapeFile*

Le format shapeFile ou fichier de forme reste le grand classique des formats vecteurs des données géographiques.

On va utiliser ici deux plugins, shp.js et leaflet-shpfile, comme dans les exemples précédents on insère dans la section <head> l'intégration des plugins :

```
<script src="leaflet/shp/dist/shp.js"></script>
<script src="leaflet/leaflet-shpfile/leaflet.shpfile.js"></script>
```

Dans le dossier data, le fichier canton.zip contient le shapeFile des cantons français compressé.

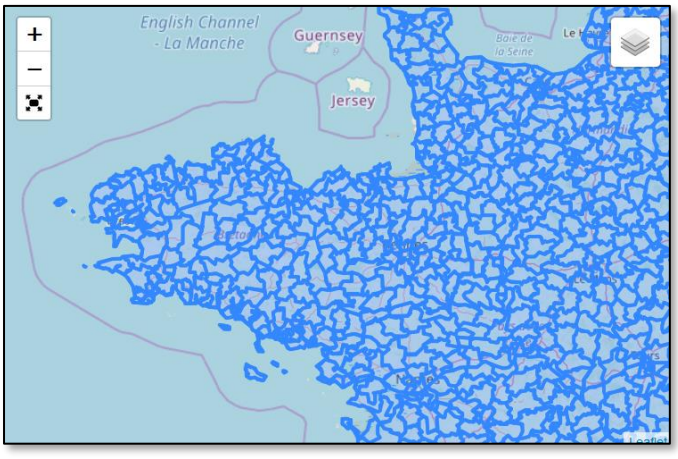


Le fait d'avoir un ZIP permet à javascript de récupérer le shapeFile en un seul téléchargement par le biais d'un seul fichier (ZIP).

Le code pour télécharger et créer la couche est le suivant :

```
var shapeLayer = new L.Shapefile("data/canton.zip");
shapeLayer.addTo(map);
```

Le résultat :



■ *Ajouter une couche Overlay Raster*

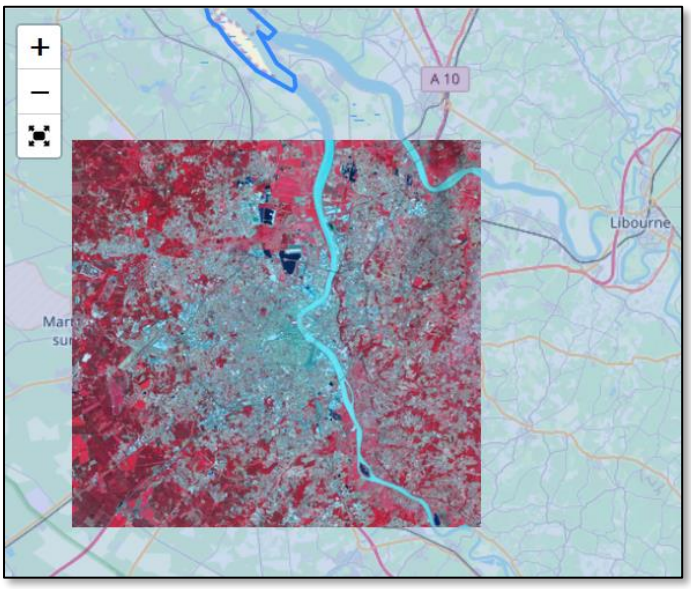
Dans le dossier data, landsat.png est une image du satellite Landsat en fausses couleurs visible / proche infra-rouge, dans la région de Bordeaux.
1442 lignes X 1367 colonnes
Etendue géographique : Ouest=-0.784355, Sud=44.70723, Est=-0.3957438, Nord=44.968463

```
satelliteLayer = new L.imageOverlay('data/landsat.png', [[44.70723, -0.784355], [44.968463, -0.3957438]]);

satelliteLayer.addTo(map);
```

La fonction L.imageOverlay() permet de créer une couche overlay raster (image).

Premier argument = url d'accès au fichier image
Second argument = Géoréférencement de l'image = étendue géographique.

Le résultat :

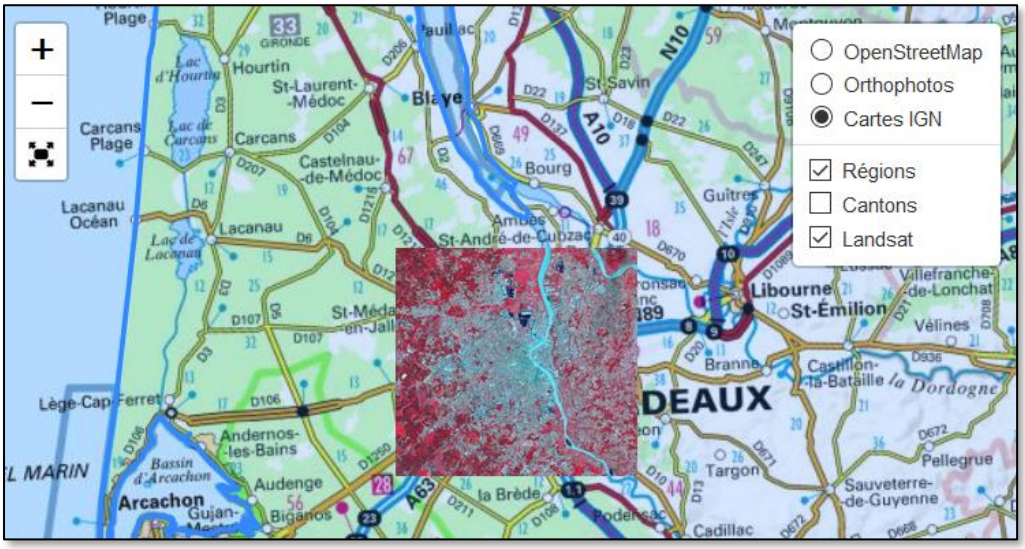


● Modifier le contrôle de couches pour gérer les Base layers et les Overlays.

L.control.layers() peut en fait accepter deux arguments, pour créer un contrôle de couches :

L.control.layers(baseLayers, overlayLayers)
baseMaps = une liste de Base Layers (selon le format vu précédemment)
overlayLayers = une liste de couches overlays.

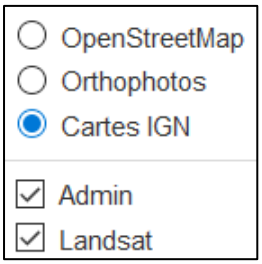
Modifier votre contrôle de couches pour qu'il intègre des base Layers et des overlays que vous avez ajoutés précédemment, comme par exemple ci-dessous :



■ *Les groupes de couches*

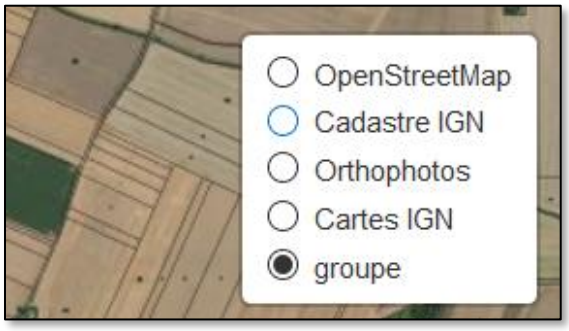
monGroupe = L.layerGroup([l1, l2, l3 …]) Permet de créer un ***groupe de couches.***

Regrouper les couches des régions et des cantons dans un groupe de couche « Admin » puis ajouter le dans le contrôle de couches à la place des deux couches de départ, de manière à pouvoir les contrôler ensemble, comme ci-dessous :



On peut aussi faire un groupe de couches avec des Base Layers.

Créer un groupe de couches avec deux Base layers : l'orthophoto à 20% d'opacité et le cadastre.



La propriété opacity (entre 0 et 1) définie l'opacité / transparence d'une couche :
couche.setOpacity(0.2) (ou couche.opacity = 0.2)

Mettre les deux couches du groupe en transparence.

■ *Ajouter de couches Overlay vecteur crées dynamiquement (en javascript)*

- Point ou Marker

L.Point(x,y) permet de créer un point ou ***L.latLng(latitude,longitude)*** créer un point d'après ses latitude et longitude, puis un construit un objet « marker » à partir de ce point et on l'ajoute :

```
var pt = L.latLng(48.84805,1.939732);
var grignon = L.marker(pt,{title:"chateau de Grignon"});
grignon.addTo(map);
map.setView(pt,15);
```

```
L.marker([48.84805,1.939732],{title:"chateau de Grignon"});  est aussi possible.
```

Le résultat :

On peut utiliser ses propres icones comme marker, avec ***L.icon***. Dans le dossier data utiliser l'icône

« chateau.png » comme icône pour le château de Grignon : 

Pour créer l'icône :

```
var monIcon = L.icon({ iconUrl: "data/chateau.png", iconSize: [40,50] });
```

Pour ajouter l'icône au marker du château de Grignon :

```
var grignon = L.marker(pt,{icon:monIcon,title:"chateau de Grignon"});
```

Le résultat :

Il existe de nombreux sites web, où l'on peut trouver des icônes gratuites, par exemple :
https://icones8.fr/icons/
http://mapkeyicons.com/
Des propriétés utiles :

Opacity (entre 0 et 1) définie le niveau de transparence du marker.
Draggable (true ou false) l'utilisateur peut déplacer le marker
Clickable (true ou false)

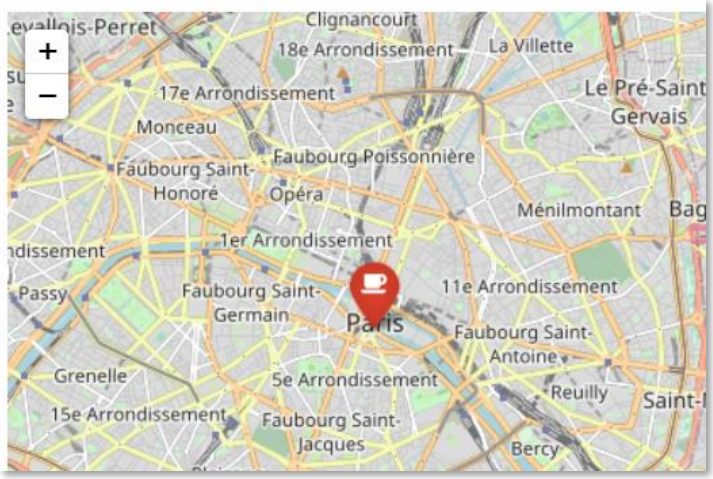
Il y au total 10 options pour L.icon : iconUrl, iconRetinaUrl, iconSize, iconAnchor, shadowUrl, shadowRetinaUrl, shadowSize, shadowAnchor, popupAnchor, className

- Il existe aussi des extensions Leaflet qui apporte une bibliothèque d'icônes, comme :
Leaflet.awesome-markers

Ajouter dans la section <head>

```
<link href="http://netdna.bootstrapcdn.com/font-awesome/4.0.0/css/font-awesome.css"
rel="stylesheet">
<link rel="stylesheet" href="leaflet/leaflet.awesome-markers/dist/leaflet.awesome-
markers.css">
```

Puis dans <body> créer un marqueur avec le code suivant :




```
var redMarker = L.AwesomeMarkers.icon({
      icon: 'coffee',
      prefix: 'fa',
      markerColor: 'red'
    });


L.marker([48.853,2.348], {icon:
redMarker}).addTo(map);
```

- Marker avec icone de type « divIcon »

L.divIcon() permet de construire un icone avec un élément html à la place d'une image comme on a vu précédemment. On peut regrouper plusieurs éléments html et leur appliquer un style CSS. On peut utiliser la possibilité de créer des formes en CSS avec le .

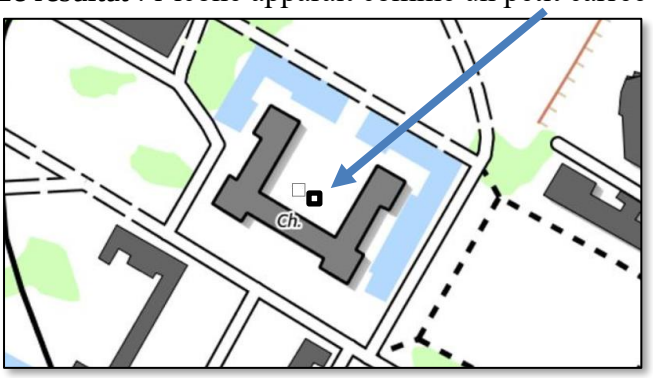
Pour créer l'icône :

```
var monIcon = L.divIcon();
```

Pour ajouter l'icône au marker du château de Grignon, comme précédemment :

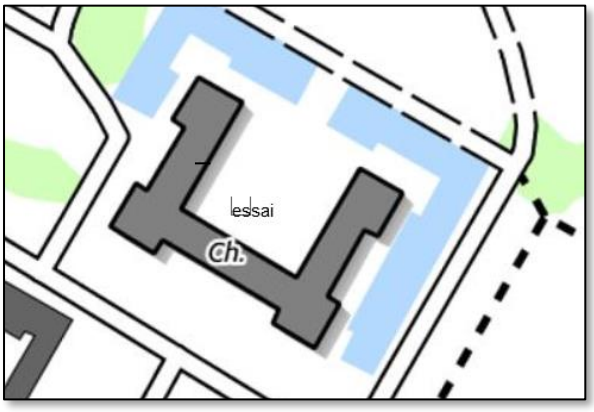
```
var grignon = L.marker(pt,{icon:monIcon,title:"chateau de Grignon"});
```

Le résultat : l'icône apparait comme un petit carrée blanc



Avec la propriété ***html***, on peut ajouter un contenu html dans la section par exemple un simple texte :

```
var monIcon = L.divIcon({html, « essai »});
```



La propriété ***iconSize*** définie la taille de l'icône. ***iconAnchor*** donne le point d'accrochage de l'icône (position lat / long) par rapport au coin supérieur gauche de l'icône.

```
var monIcon = L.divIcon({html: "essai", iconSize:[100,100],iconAnchor: [100,100]});
```

Le résultat : un carré de 100x100 et le point d'accrochage en bas à droite (100x100 par rapport au coin haut gauche)

La propriété **className**, permet d'utiliser un style CSS.

Dans la section <head> et dans une section <style> définir un style css « .css-icon » où l'on fixe la couleur de l'arrière-plan.

```
<style>
.css-icon {
  background: lightblue;
}
</style>
```

Et l'appeler dans L.divIcon :

```
var monIcon = L.divIcon({html: "essai", iconSize:[100,100],iconAnchor:
[100,100],className: "css-icon"});
```

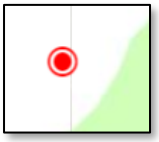
divIcon basique seulement avec des caractères

Le plus simple est encore d'utiliser des caractères dans le contenu html de L.divIcon() pour symboliser les points, on pourra ainsi faire varier facilement le style par la taille et la couleur (font-size, color). On peut utiliser des **caractères standards** : + x o * … ainsi que des **caractères semi-graphiques** : ■ (\u220e) ⊗ (\u2297) ⊕ (\u2a01) ⚬ (\u2218) ◉ (\u25c9) ◈ (\u25c6) …

https://fr.wikipedia.org/wiki/Table_des_caract%C3%A8res_Unicode/U25A0

```
.css-point{
  color: red;
  position : absolute ;
  line-height: 0.5em;
  left:-0.1em;
  font-size:14px;
  background: transparent;
}

L.divIcon({html: "\u25c9" , className: "css-point"});
```

Unité em = taille de la police de caractère = 1 em = la hauteur du caractère.
Le résultat :



Voyons maintenant comme faire des formes géométriques plus complexes avec le L.divIcon.

Carré et Rectangle c'est la forme de base du Div, on peut jouer sur la taille.

On peut aussi, en particulier pour des objets objets plus complexe, insérer une ou plusieurs div dans la propriété html de L.divIcon() comme ci-dessous :

Un triangle :

```
.css-i {
 background: transparent;
 text-align: center;
}

.css-icon {
        width :0 ;
        height :0 ;
        border-left: 50px solid transparent;
        border-right: 50px solid transparent;
        border-top: 100px solid blue;
        opacity: 0.8;
}
```

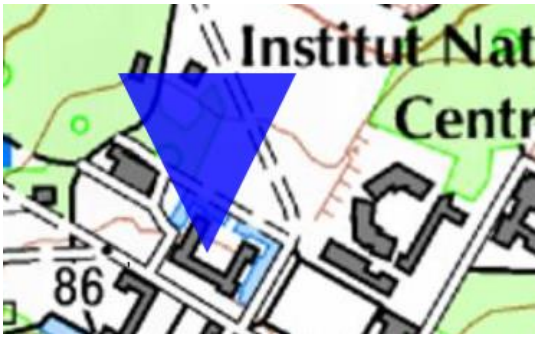
```
var monIcon = L.divIcon({html:"<div class='css-icon'></div>" ,className : "css-i",iconAnchor:[50,100]});
```

On fixe la largeur du côté gauche du div à 50 pixels (transparent), le droit à 50 px (transparent), le haut à 100 pixels et pas de côté bas. C'est la couleur du côté haut qui donne la couleur du triangle.
iconAnchor pour décaler le triangle : pour que le sommet bas pointe sur le point des coordonnées.

css-i sert à « effacer » le petit carré blanc par défaut.

 Attention la taille finale du div est augmentée par les épaisseurs des bords, ainsi si on veut comme dans l'exemple du triangle un div 100 x 100, le plus simple est de mettre à zéro width et height et de s'arranger pour que border-left + border-right = 100 et border-top=100.

Le résultat :

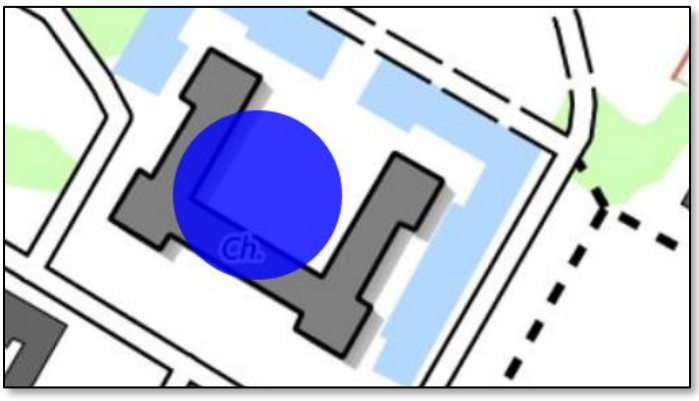


Un cercle

```
.css-icon {
        background: blue;
        border-radius: 50%;
        opacity: 0.8;
}
```

Border-radius permet d'arrondir les coins d'un carré, en fixant la moitié du côté du carré on obtient un cercle.

Le résultat :



```
var monIcon = L.divIcon({ html:"<div class='css-i1'>A</div><div class='css-i2'></div>",iconSize:[100,80],iconAnchor:[50,80],className:'css-i'});
```

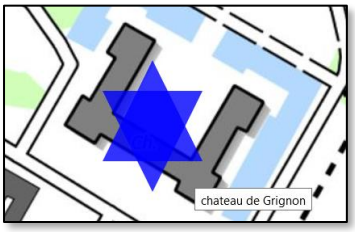
On peut créer des icônes plus complexes en utilisant plusieurs comme ce qui suit :

Une étoile composée de deux « triangle »

```
.css-i1 {
        width:0;
        height:0;
        border-left: 50px solid transparent;
        border-right: 50px solid transparent;
        border-bottom: 100px solid blue;
   opacity: 0.8;
}
.css-i2 {
        width:0;
        height:0;
        border-left: 50px solid transparent;
        border-right: 50px solid transparent;
        border-top: 100px solid blue;
   opacity: 0.8;
        position: absolute;
        top: 30px;
}
```

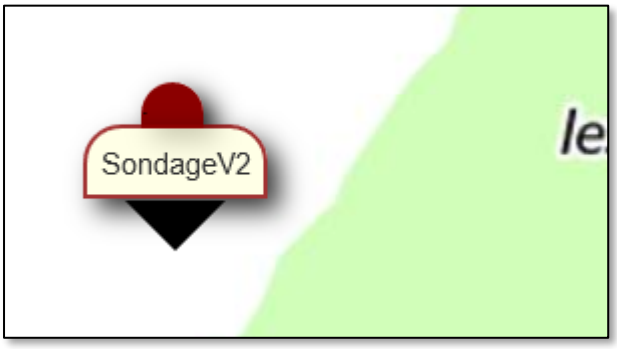
 Position top 30 px pour décaler le second triangle vers le haut.

```
var monIcon = L.divIcon({html:"<div class='css-i1'></div><div class='css-i2'></div>",className:"css-i",iconAnchor:[50,50]});
```



On verra dans le point 5, comment adapter la taille de symbole à l'échelle d'affichage (zoom).

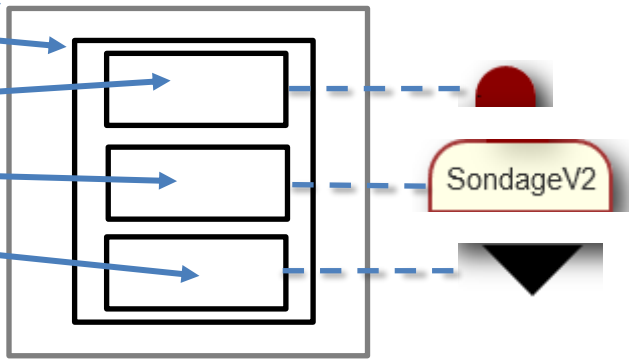
Un exemple d'icone divIcon plus complexe, composée de 3 parties :



Les boites avec leur style css correspondent sont structurées comme ci-dessous :

L.divIcon(className : 'css-fond'

```
<div class='css-i'>
<div class='css-i0'>
<div class='css-i1'>
<div class='css-i2'>
```

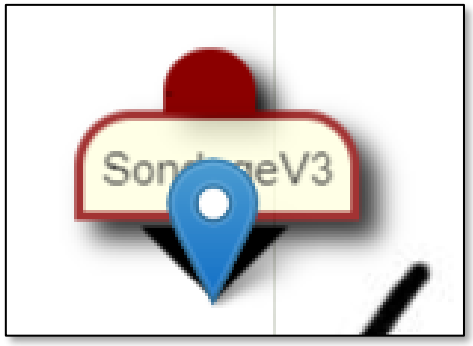


Le contenu pour la propriété « html » de L.divIcon :

```
contenu="<div class='css-i'><div class='css-i0'>.</div><div class='css-
i1'>" +"Sondage"+feature.properties.title + "</div><div class='css-
i2'></div></div>"

L.divIcon({html:contenu,className:'css-fond'});
```

Les styles css :

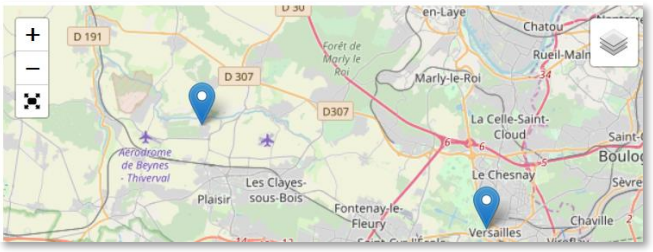
```
.css-fond{
      background: transparent;
}
.css-i {
      display:table-cell;
      align:center;
      vertical-align: middle;
}
.css-i0{
  position: relative;
  left: -7px;
  top:-64px;
  width:25px;
  heigh:20px;
  background-color: darkred;
  border-bottom:10px;
  border-top-left-radius: 15px;
  border-top-right-radius: 15px;
  box-shadow: 5px 5px 10px;
}
.css-i1 {
  position: relative;
  left:-40%;
  height:16px;
  top:-65px;
  background-color: lightyellow;
  border-top-left-radius: 15px;
  border-top-right-radius: 15px;
  border-width: 2px;
  border-style: solid;
  border-color: darkred;
  text-align: center;
  padding: 5px;
  opacity:0.8;
  box-shadow: 5px 5px 10px;
  bl: 10px;
}
.css-i2 {
align-self: center;
width :0 ;
height :0 ;
height:20px;
border-left: 20px solid transparent;
border-right: 20px solid transparent;
border-top: 20px solid black;
position: relative;
left: -13px;
top:-64px;
}
```



⚠ Attention lors de la construction de ce genre d'icone il n'est pas évident de bien gérer le centrage et la taille des éléments, afficher un marker standard sur votre icône pour vérifier qu'elles pointent bien au même endroit.

- Créer une couche de points en format geoJSON.

Créer une couche geoJSON « chateauxLayer » avec le château de Grignon auquel on ajoute ensuite le château de Versailles comme ci-dessous :




```
var chateauGrignon = [
 { "type": "Feature",
   "geometry": {
        "type": "Point",
        "coordinates": [1.939732,48.84805]
    },
    "properties": {
        "name": "Grignon",
        "title": "Grignon" }
}];

var chateauxLayer = L.geoJson(chateauGrignon);
chateauxLayer.addTo(map);

var chateauVersailles = [
 { "type": "Feature",
   "geometry": {
        "type": "Point",
        "coordinates": [2.121362,48.804581]
    },
    "properties": {
        "name": "Versailles",
        "title": "Versailles" }
}];

chateauxLayer.addData(chateauVersailles);
```

Remarquez la fonction **addData** qui permet d'ajouter un feature à une couche geoJSON existante.

Les couches en format geoJSON ont souvent plus de possibilités en termes de manipulation des objets, comme ici pour ajouter un nouveau château.

Les objets Leaflet ont généralement la méthode « .toGeoJson() » pour être convertie en GeoJson.

- Les autres géométries que les points :

`L.polyline, L.polygon,  L.rectangle et L.circle.`

Créer un triangle (polygon) délimitant le quartier du site de AgroParisTech au 16 rue Claude Bernard à Paris :

```
var AgroParisTech = L.polygon([
   [48.839864,2.347293],
   [48.840370,2.349139],
   [48.839240,2.348892]
]);

AgroParisTech.addTo(map);
```

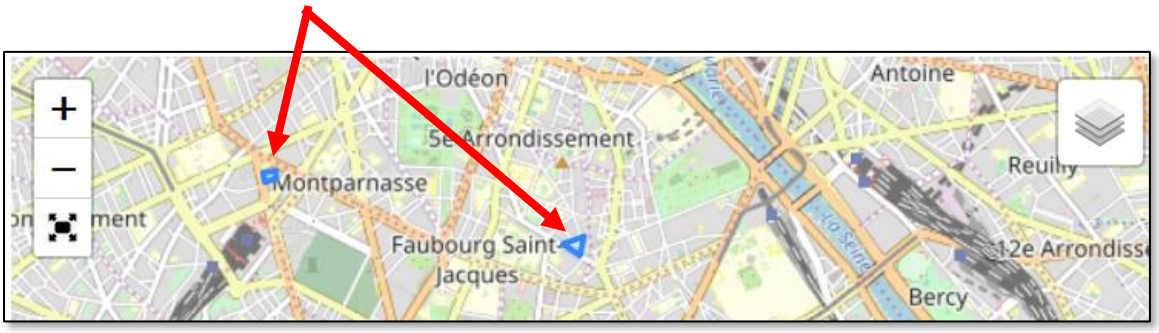
Le résultat :

On peut aussi créer plusieurs polygones dans une même couche.

Modifier la couche « AgroParisTech » comme suit, pour intégrer les deux sites de la rue Claude Bernard et de l'avenue du Maine de AgroParisTech.

```
var AgroParisTech = L.polygon([
 [
  [48.839864,2.347293],
  [48.840370,2.349139],
  [48.839240,2.348892]
 ],
 [
  [48.844068,2.320466],
  [48.844184,2.321695],
  [48.843697,2.321663],
  [48.843528,2.320692]
 ]
]);
```

Le résultat : les deux sites



Les deux mêmes polygones sous la forme d'une couche geoJSON

```
var agro = [
 { "type": "Feature",
   "geometry": {
        "type": "Polygon",
        "coordinates": [
                        [
                          [2.347293,48.839864],
                          [2.349139,48.840370],
                          [2.348892,48.839240]
                        ]
                       ] },
    "properties": {
        "name": "Claude Bernard",
        "title": "Claude Bernard" }
},
 { "type": "Feature",
   "geometry": {
        "type": "Polygon",
        "coordinates": [
                        [
                          [2.320466,48.844068],
                          [2.321695,48.844184],
                          [2.321663,48.843697],
                          [2.320692,48.843528]
                        ]
                       ] },
    "properties": {
        "name": "Av. du Maine",
        "title": "Av. du Maine" }
}];
var AgroParisTech = L.geoJson(agro);

AgroParisTech.addTo(map);
```

4. Style et légende des couches

La propriété « style » permet de personnaliser le style d'affichage de deux façons différentes :

- En passant un objet style : couleur, transparence …
- En passant une fonction de style qui peut servir à faire des légendes automatiques.

■ *Style statique*

Définir le style de la couche, lors de la création des régions, pour avoir : un contour rouge et un fond bleu avec une opacité de 50% et une épaisseur de trait de 5, comme ci-dessous :

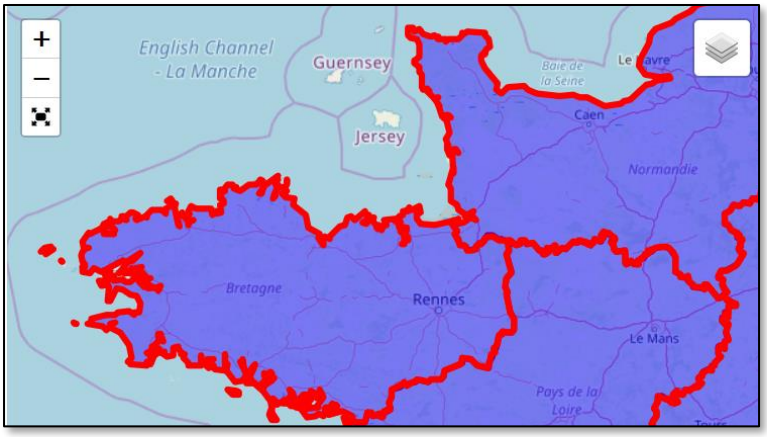
```
var myStyle = {
 "color": "red",
 "weight": 5,
 "opacity": 1.0,
 "fillColor": "blue",
 "fillOpacity": 0.5
};

var geojsonLayer = new L.GeoJSON(geoj,{style: myStyle});
```

On peut aussi passer une seule propriété directement : {"fillColor":"gray"} par exemple

Noter les propriétés graphiques du style :

- color = couleur du contour
- weight = épaisseur du trait
- opacity = opacité du contour
- fillColor = couleur de remplissage du polygone
- fillOpacity = opacité du remplissage du polygone

Le résultat :



On peut aussi changer le style d'une couche existante à l'aide de la **fonction setStyle** :

geojsonLayer.setStyle({"fillColor":"gray"}); pour changer ou plusieurs propriétés
geojsonLayer.setStyle({myStyle}); redonner une variable de style complète

Attention setStyle sur les couches chargées avec L.Shapefile semble un peu beuguée :

Sur l'exemple précédent des cantons :

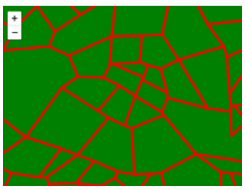
```
var shapeLayer = new L.Shapefile("data/canton.zip");
shapeLayer.setStyle({"fillColor":"green"});
shapeLayer.addTo(map);
```

Ne marche pas le fond n'a pas changé 

Passer par une variable de style et changer la variable sans utiliser setStyle :

```
var myStyle = {
 "color": "red",
 "weight": 5,
 "opacity": 1.0,
 "fillColor": "blue",
 "fillOpacity": 0.5
};

var shapeLayer = new L.Shapefile("data/canton.zip",{style :myStyle});
myStyle["fillColor"]="green";
shapeLayer.addTo(map);
```

Le changement marche aussi après avoir fait le addTo(map)

Marche le fond est devenu vert 

Si le changement de style est fait à l'intérieur d'une fonction appelée par un évènement (click …) il faut alors bien utiliser setStyle :

```
function GoTo(){
    shapeLayer.setStyle({"fillColor":'green'});
};
```

■ *Style dynamique, fonctions de style : style et pointToLayer*

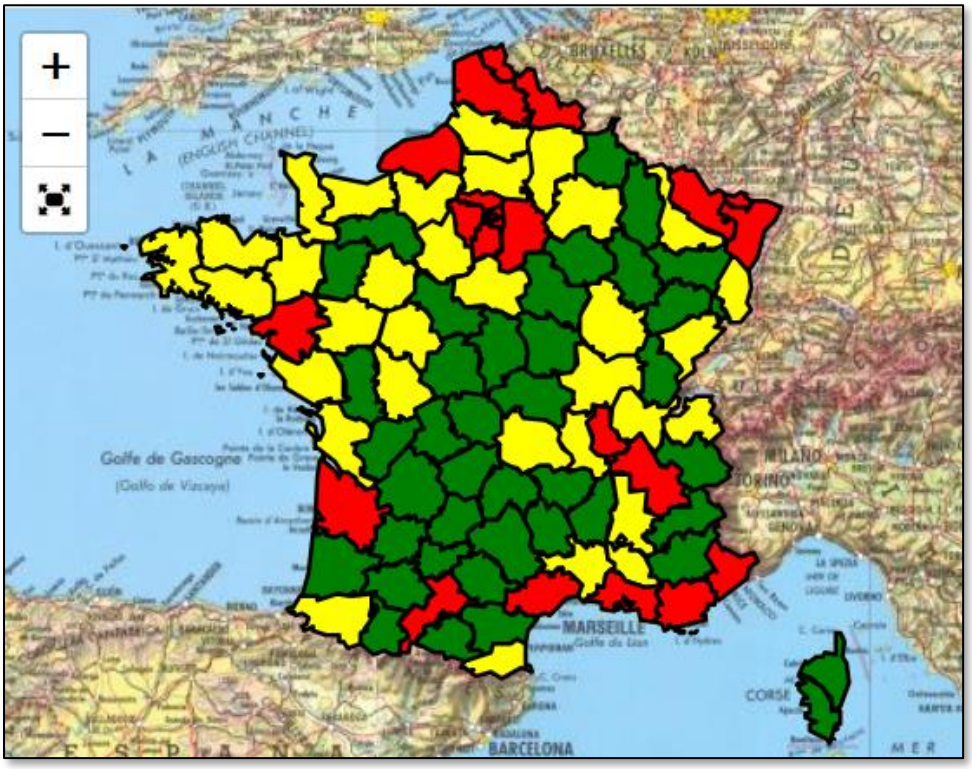
Ajouter la couche des départements en format shapeFile (departement.zip) depuis le dossier « data ».

Le résultat :

Créer une fonction de style qui affiche la population des départements, en 3 classes selon un dégradé de couleurs. Le champ « POP » contient la population du département.

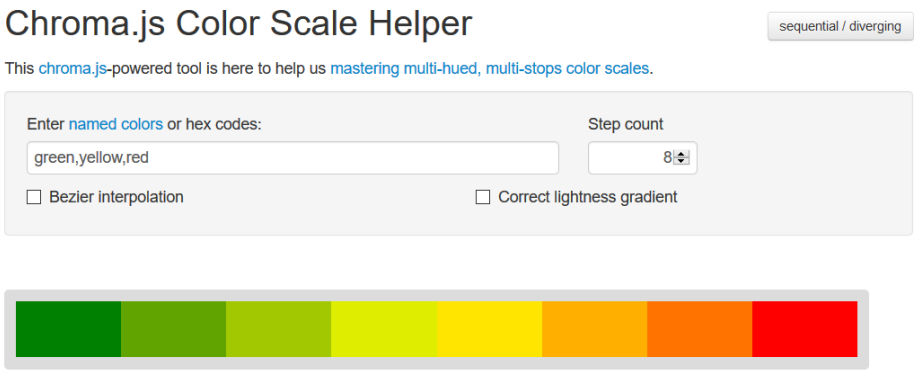
```
function styleDep(feature) {

  if (feature.properties.POP < 400000) {
   couleur = "green";
  }
  else if (feature.properties.POP < 950000) {
  couleur = "yellow";
  }
  else couleur = "red";

    return {
        fillColor: couleur,
        weight: 2,
        opacity: 1,
        color: 'black',
        fillOpacity: 1
    };
}

var dep = new L.Shapefile("data/departement.zip",{style:styleDep});
dep.addTo(map);
```

Le résultat :



Créer la même carte mais avec 8 classes, avec un gradient de couleurs continues du vert > jaune > rouge.

Aidez-vous du site : https://gka.github.io/palettes qui permet de générer des palettes.

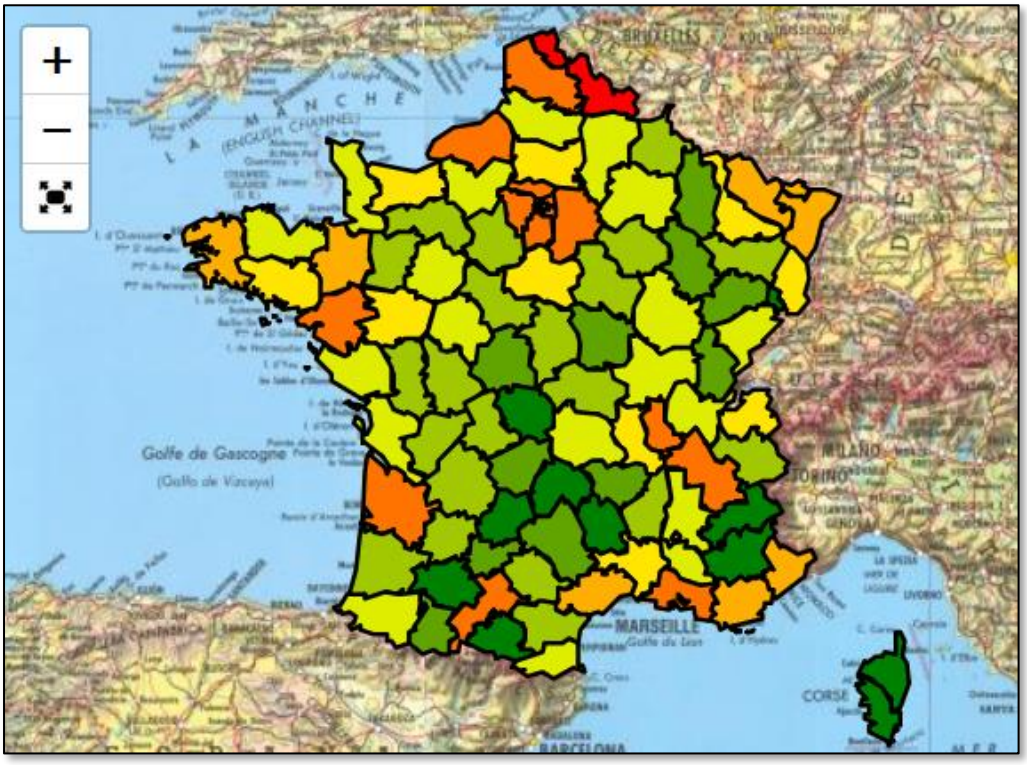


Copier / coller la ligne suivante pour en faire un tableau javascript contenant votre palette de couleurs :

```
'#008000','#61a400','#a1c800','#dfed00','#ffe500','#ffaf00','#ff7400','#ff0000'
```

```
var palette = ['#008000','#61a400','#a1c800','#dfed00','#ffe500','#ffaf00','#ff7400','#ff0000'];
```

Le résultat :



 Créer la couche des aéroports du monde à partir du shapeFile « ne_10m_airports.zip » en utilisant comme symbole « avion.png »  et « fusee.png »  en fonction du type d'aéroport) (champ TYPE) Vous distinguerez : major (les plus gros), mid (moyen), small, spaceport et autres.

 La propriété **pointToLayer** dans un constructeur de couche de points, permet d'associer une fonction (fcIcon) aux points de façon à pouvoir, par exemple, adapter les icones du marker : {pointToLayer:fcIcon}

pointToLayer est l'équivalent de la fonction style pour les géométries ponctuelles.

```
function style_aero(feature,latlng){
switch(feature.properties.type) {
  case "major":
    return L.marker(latlng,{icon:L.icon({ iconUrl: "data/avion.png", iconSize: [35,35]})});
    break;
  case "mid":
    return L.marker(latlng,{icon:L.icon({ iconUrl: "data/avion.png", iconSize: [20,20]})});
    break;
  case "small":
    return L.marker(latlng,{icon:L.icon({ iconUrl: "data/avion.png", iconSize: [10,10]})});
    break;
  case "spaceport":
    return L.marker(latlng,{icon:L.icon({ iconUrl: "data/fusee.png", iconSize: [35,35]})});
    break;
  default:
    return L.circleMarker(latlng);
}
}


var aero = new L.Shapefile("data/ne_10m_airports.zip",{pointToLayer:style_aero});

aero.addTo(map);
```

Le résultat :

■ *Légende : le contrôle de légende*

Ajouter une légende à la carte de la population des départements en 3 classes.

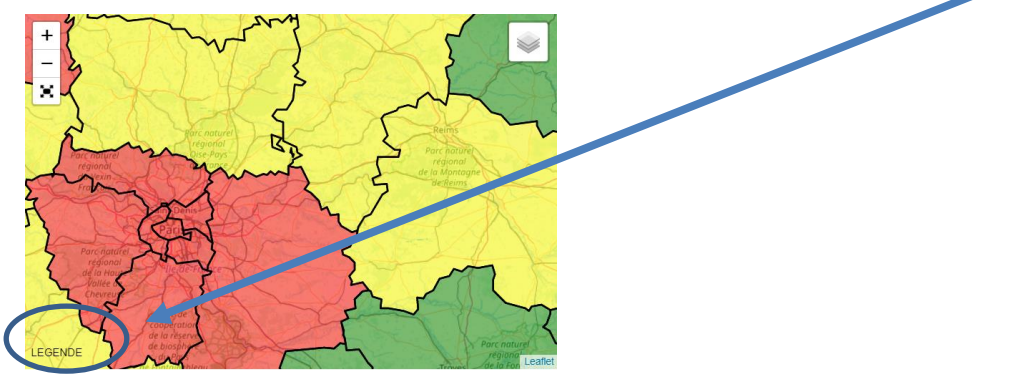
```
legend = L.control({position: 'bottomleft'});

legend.onAdd = function (map) {
    var div = L.DomUtil.create('div', 'legend');
    div.innerHTML = 'LEGENDE';
   return div;
};

legend.addTo(map);
```

legend.addTo(map) ou map.addControl(legend) map.removeControl(legend) pour enlever une légende

On créer un nouveau contrôle positionné en bas à gauche.

Legend.onAdd = une fonction qui se déclenche et dessine la légende quand on ajoute le contrôle à la carte avec legend.addTo(map) ;

div.innerHTML = le contenu html qui est affiché, pour le moment le titre « LEGENDE »



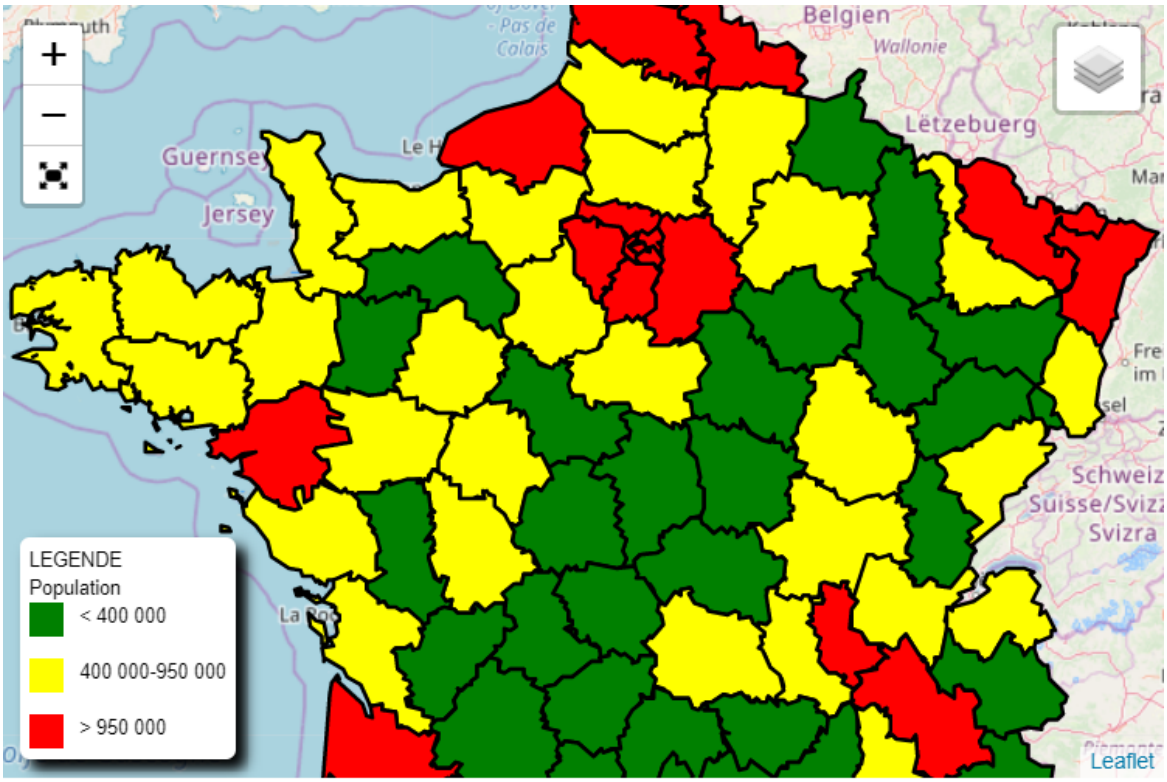
On va créer une liste de 3 entrées (les 3 classes d'affichages) en html :

```
legend.onAdd = function (map) {
 var div = L.DomUtil.create('div', 'legend');
 div.innerHTML = 'LEGENDE<br>Population';
 div.innerHTML += "<i style='background-color:green'> </i>< 400 000<br><br>";
 div.innerHTML += "<i style='background-color:yellow'> </i> 400 000 - 950 000<br><br>";
 div.innerHTML += "<i style='background-color:red'> </i> > 950 000";
 return div;
};
```

Un peu de style css pour la mise en forme pour le bloc légende (.legend) et pour les entrées de la liste (.legend i) :

```
.legend{
background-color:white;
box-shadow: 5px 5px 10px black;
border-radius: 5px;
padding:5px;
}

.legend i {
    width: 18px;
    height: 18px;
    float: left;
    margin-right: 8px;
}
```

Le résultat :



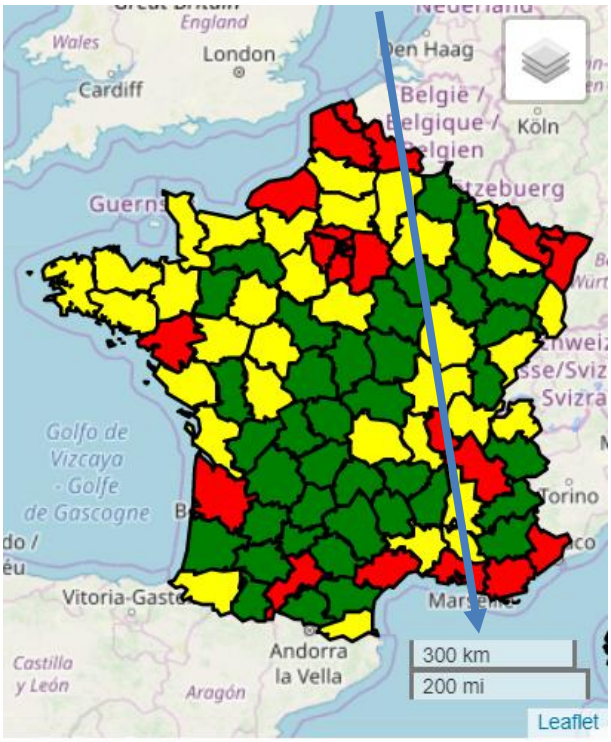
■ Barre d'échelle

Le contôle de barre d'échelle standard de Leaflet :

```
bar = L.control.scale({position: 'bottomright'});
bar.addTo(map);
```

Résultat :

La barre d'échelle



Pour avoir que le système métrique :

bar = L.control.scale({position: 'bottomright',imperial:false});

Avec le plugin betterscale

(les scripts de betterscale ont été téléchargés depuis le site de Leaflet dans notre dossier leaflet)

intégrer le plugin (.js et .css):

```
<link rel="stylesheet" href="leaflet/leaflet-betterscale/L.Control.BetterScale.css" />
<script src="leaflet/leaflet-betterscale/L.Control.BetterScale.js"></script>
```

Puis dans votre script ajouter le contrôle de barre d'échelle :

```
bar = L.control.betterscale({position: 'bottomright',imperial:false,metric:true});
bar.addTo(map);
```

■ *Etiquettes*

● Afficher des étiquettes : .bindTooltip()

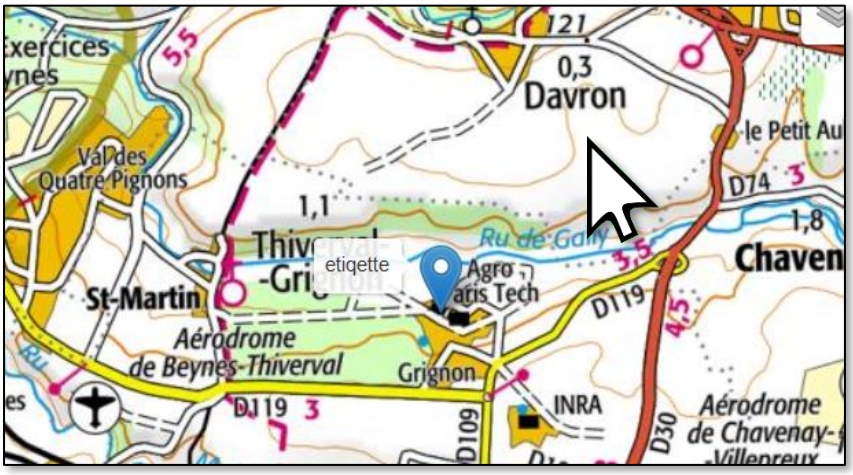
On a vu précédemment sur le marker du château de Grignon, quand on approche le curseur du marker un message s'affiche : c'est la propriété « titre » du Marker.

- La fonction .bindTooltip() permet d'ajouter une étiquette en plus.

```
var grignon = L.marker(pt,{title:"chateau de Grignon"}).bindTooltip("etiquette");
```

- Rendre l'étiquette permanente, même quand le curseur n'est pas sur le Marker :

```
var grignon = L.marker(pt,{title:"chateau de
Grignon"}).bindTooltip("etiqette",{permanent:true});
```



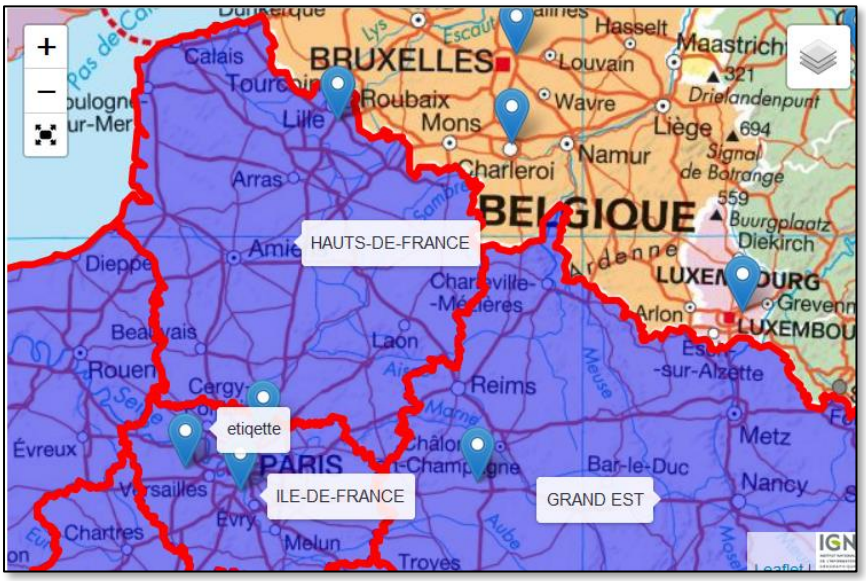
- Crée une étiquette automatique à partir d'une propriété (champ) : onEachFeature

onEachFeature est une option qui permet d'appeler une fonction lors de la construction des objets (feature) de la couche, pour faire un traitement spécifique objet par objet.

Ajouter le nom des étiquettes correspondant au nom des régions (champ « NOM_REG »)

```
function featureFct(feature,layer){
 layer.bindTooltip(feature.properties.NOM_REG,{permanent:true});
}

var geojsonLayer = new
L.GeoJSON(readGeojson("data/regions.geojson"),{style:
myStyle,onEachFeature: featureFct});
```

Le résultat :

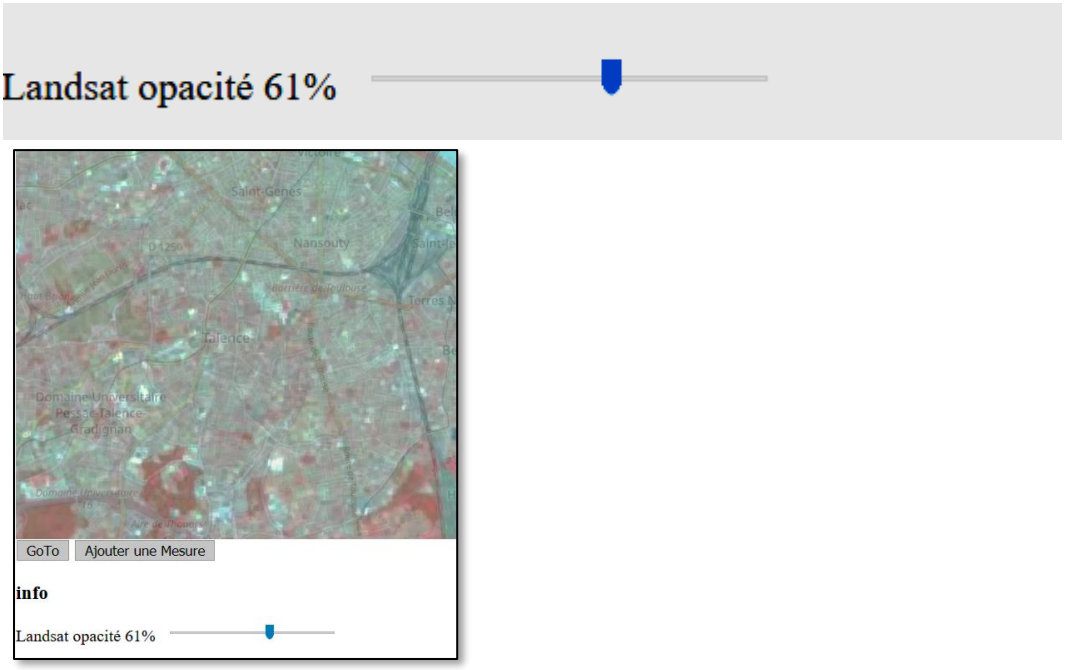


■ *Effet de transparence d'une couche*

Pour les vecteurs, la propriété fillOpacity (entre 0 et 1) définie l'opacité / transparence d'une couche. (opacity pour le contour et fillOpacity pour le remplissage)
La fonction setStyle() peut être utiliser pour changer la transparence ou plus globalement le style d'une couche.
Pour les raster c'est la propriété opacity, et on peut utiliser la fonction .setOpacity() pour la changer, à la création de la Layer avec L.imageOverlay mettre bien en option {opacity:1.0} pour initialiser la transparence à zéro.

Ajouter en bas de page un contrôle de type slider, pour contrôler interactivement le niveau d'opacité de la couche Landsat.

Vous associerez une fonction javascript aux changements du slider, cette fonction changera le niveau d'opacité de la couche landsat et actualisera le texte (Landsat opacité 61%) devant le slider.




```
<p id="opacite"  style="display:inline">Landsat opacité 100%</p>
```

Display :inline évite un saut de ligne dans la mise en page

Code html pour un slider :

```
<input type="range" min="0" max="100" value="100" class="slider" id="slider"
onchange="updateSlider(this.value)">
```

Onchange= updateSlider associe la fonction javascript updateSlider à chaque fois que l'utilisateur change la position du curseur.

```
var satLayer = new L.imageOverlay('data/landsat.png', [[44.70723,     -0.784355], [44.968463, -
0.3957438]],{opacity:1.0});

satLayer.addTo(map);

function updateSlider(value){
  satLayer.setOpacity(value / 100);
}
```

5. Interactions avec la carte

■ *Boite de dialogue d'information en cliquant sur un objet*

Afficher une boite de dialogue « popup » d'information en cliquant sur un objet

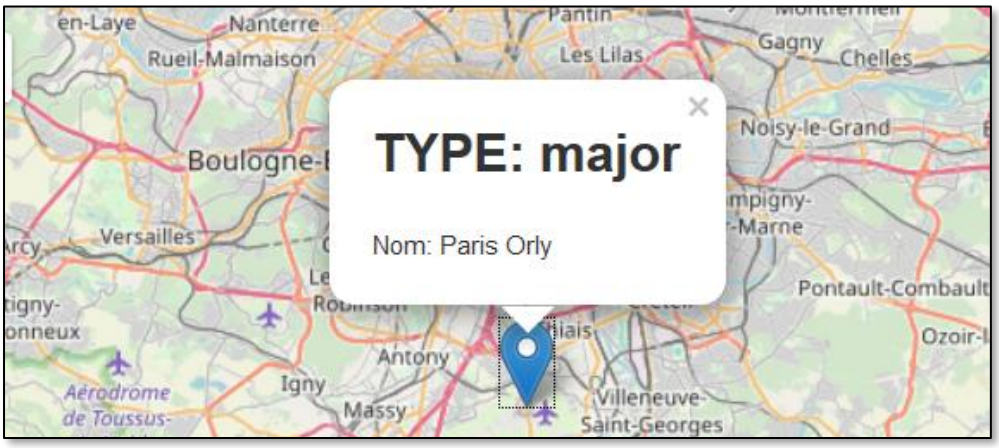
Les informations seront extraites des « properties » (initialement les champs de la couche SIG).

 Faire une boite de dialogue d'information pour les aéroports en affichant les champs « type » et « name »

On utilise ici aussi la propriété **onEachFeature**.

```
function popupFct(feature,layer){
layer.bindPopup('<h1>TYPE: '+feature.properties.type+'</h1><p>Nom:
'+feature.properties.name+'</p>');
}
var ap = new L.Shapefile("data/ne_10m_airports.zip",{onEachFeature:popupFct});
```

Le résultat :



■ *Centrer la vue sur une position demandée*

Centrer et zoomer sur une position demandée par l'utilisateur

On ajoute un bouton « GoTo » en bas de page qui appel une fonction javascript qui va centrer la carte sur les coordonnées latitude longitude données par l'utilisateur, le code HTML du bouton est le suivant :

```
<button onclick="GoTo()">GoTo</button>
```

A mettre après le script dans le code html (pas dans le javascript)

Bouton « GoTo »

On va programmer la fonction GoTo dans la section <script>

```
<script>

function GoTo(){
 alert("ok");
}

</script>
```

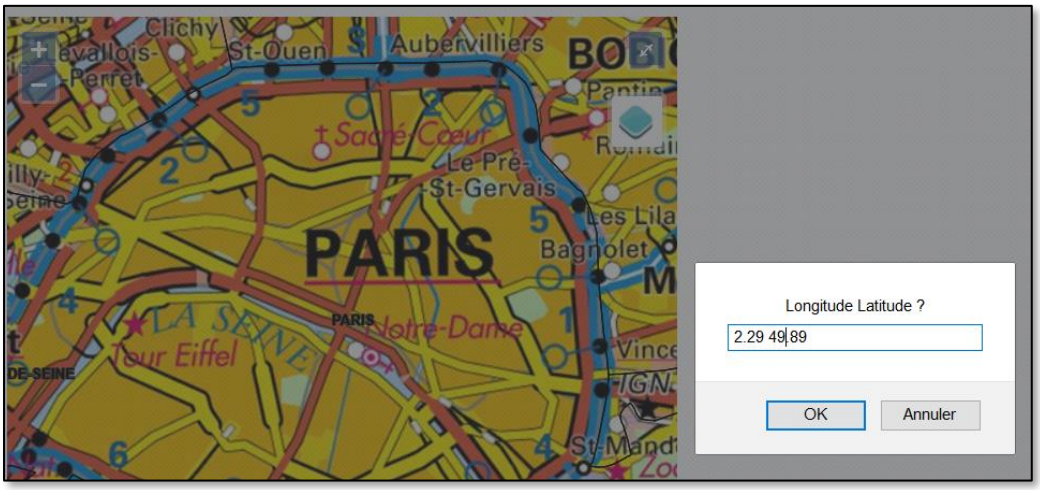
La fonction alert() affiche une boite de dialogue avec un message, pour tester que l'appel de la fonction marche.

La fonction prompt() permet de demander des informations elle renvoie la chaine de caractères entrée par l'utilisateur.

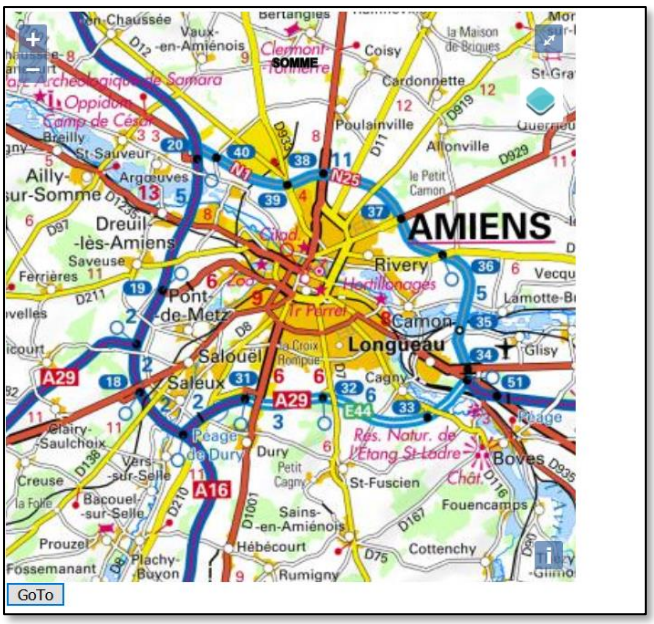
```
function GoTo(){
 position = prompt("Longitude Latitude ?").split(' ');

 lon = parseFloat(position[0]);
 lat = parseFloat(position[1]);

 map.setView([lat,lon],12);

}
```

La fonction split() permet d'éclater une chaine en mots dans un tableau (position) séparateur = espace
parseFloat() converti en un réel

Tester votre code sur la ville d'Amiens : longitude = 2.29 latitude=49.89



Résultat :



■ *La gestion des évènements*

La classe L.Map supporte 34 événements. On peut associer des fonctions à ces événements pour traiter les interactions de l'utilisateur avec la carte, par exemple un clic sur la carte. Pour traiter ces événements, on doit connaitre le type d'événement qui nous intéresse, son nom et les données qu'il renvoi.

On peut donc associer des évènements à d'autres objets que « map » tels que les couches par exemple.

Voici la liste des principaux événements de l'objet « Map » :

	focus	
	blur	zoomlevelschange
	preclick	
	load	resize
		autopanstart
	unload	layeradd
click	viewreset	layerremove
dblclick	movestart	baselayerchange
mousedown	move	overlayadd
mouseup	moveend	overlayremove
mouseover	dragstart	locationfound
mouseout	drag	locationerror
	dragend	popupopen
mousemove	zoomstart	
		popupclose
contextmenu	zoomend	

Ouvrir une boîte de dialogue quand l'utilisateur clic sur la carte.

```
map.on("click", function() { alert("clic sur la carte!"); })
```

Le type de données retourné peut varier selon le type d'événement : Event, MouseEvent, LayerEvent …
Par exemple pour un MouseEvent, une des propriétés retournée est **latlng** = le lieu du clic
Remarquer la syntaxe : la fonction (sans nom) est directement définie dans l'appel de la fonction map.on(), on peut aussi d'abord créer une fonction avec un nom puis l'affecter à l'évènement avec map.on (peut être plus lisible).

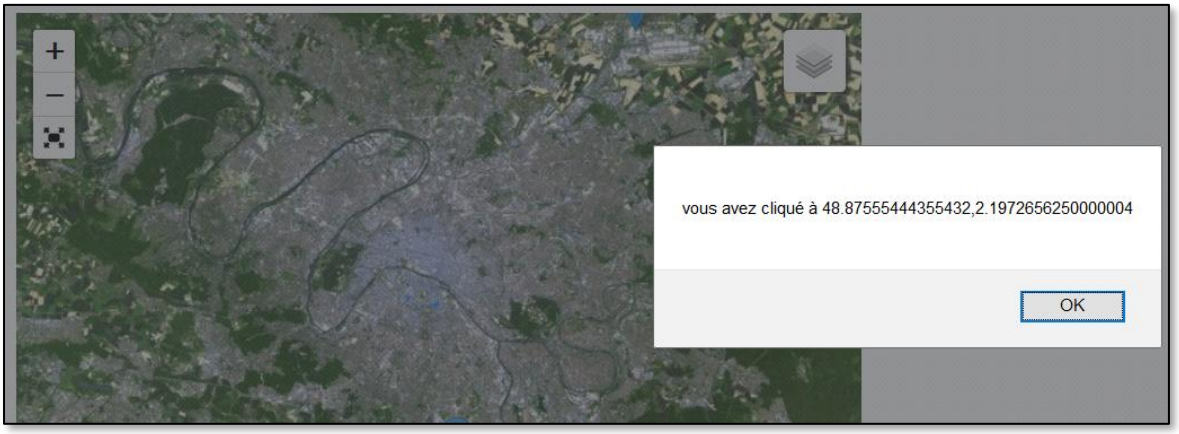
map.on(« click » active l'association de l'évènement « click » avec la fonction créée

map.off(« click » pour désactiver

Modifier la gestion d'événement précédente pour écrire dans la boite de dialogue les latitude / longitude du point cliqué.

```
map.on("click", function(e) {

var lat = e.latlng.lat;
var lng = e.latlng.lng;
alert("vous avez cliqué à " + lat + "," + lng);

})
```

La variable e contient les données de l'événement.
Le résultat :



Faire une fonction associée à un évènement « click » de la couche des départements qui affiche en dessous de votre carte le nom du département sur lequel l'utilisateur à cliquer.

Après la partie <script> de Leaflet ajouter à la page un tag c'est un conteneur générique en html
On lui donne un identifiant : id= « message »

```
</script>
<div id="message">texte</div>
</body>

</html>
```

```
function departement_event(e) {
  document.getElementById("message").innerHTML=e.layer.feature.properties.NOM;
}
```

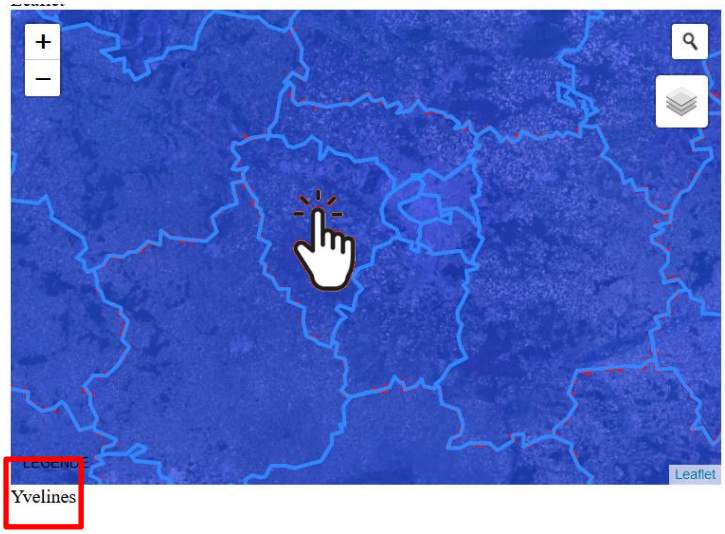
En javascript document.getElementById("message") récupère un objet de la page HTML grâce à son identifiant et innerHTML permet de le modifier avec du code HTML (ici un simple texte)

```
On associe cette fonction à l'évènement click souris pour la couche département

departement.on('click',departement_event );
```

On peut donc associer des évènements à des couches pas seulement à « map ».

Avant de cliquer remarquer « texte » sous la carte

Ensuite quand on clique sur un département, son nom apparait sous la carte (dans le)



Faire la même fonction mais associée à un évènement de survol par la souris « mouseover » de la couche des départements qui affiche en dessous de votre carte le nom du département.

Faire un **symbole** pour le château de Grignon sous la forme d'un cercle dont le rayon dépend du niveau de zoom : **la taille augmente quand on de zoom**.

Il faut une fonction qui capture l'évènement « fin de zoom » : **zoomend**

On définit un style de départ pour notre cercle :

```
.css-rond{
        width: 0px;
        height: 0px;
        background: blue;
        border-bottom: 50px solid blue;
        border-top: 50px solid blue;
        border-left: 50px solid blue;
        border-right: 50px solid blue;
        border-radius: 50%;
        position: absolute;
        top: 50px;
        left: 0px;
        opacity: 0.8;
}
```

La création du marker (L.divIcon)

```
var pt = L.latLng(48.84805,1.939732);
var monIcon = L.divIcon({className:"css-rond"});
var grignon = L.marker(pt,{icon:monIcon,title:"Grignon"});
grignon.addTo(map);
```

La fonction va lire le style CSS et le changer en fonction du niveau de zoom, il y a deux possibilités : changer les propriétés des objets ou du style lui-même :

-- En changeant les propriétés de tous les objets qui possèdent le style css « .css-rond »

```
map.on('zoomend', function(e) {

    var zoom = map.getZoom();
    var elements = document.querySelectorAll(".css-rond");

    taille = Math.round( 10 / Math.log(zoom));
    for (i = 0; i < elements.length; i++) {
        elements[i].style["border-left"] = String(taille) + "px solid blue" ;
        elements[i].style["border-right"] = String(taille) + "px solid blue";
        elements[i].style["border-top"] = String(taille) + "px solid blue";
        elements[i].style["border-bottom"] = String(taille)+ "px solid blue" ;
        elements[i].style["top"] = "-"+String(taille)+ "px" ;
        elements[i].style["left"] = "-"+String(taille)+ "px" ;
    }
});
```

On modifie par javascript pour adapter la taille du symbole.
On utilise les largeurs de bord comme vu précédemment, left et top pour centrer le symbole quand la taille change.

- En changeant la définition du style lui même

```
   map.on('zoomend', function(e) {

    var zoom = map.getZoom();

    var styleSheets = window.document.styleSheets;

    for(var st=0; st<styleSheets.length; st++) {
     var classes = styleSheets[st].cssRules;
     for (var r = 0; r < classes.length; r++) {
       if(classes[r].selectorText  == ".css-rond") {
        taille = Math.round( 10 / Math.log(zoom));

        classes[r].style["border-left"] = String(taille) + "px solid blue" ;
        classes[r].style["border-right"] = String(taille) + "px solid blue";
        classes[r].style["border-top"] = String(taille) + "px solid blue";
        classes[r].style["border-bottom"] = String(taille)+ "px solid blue" ;
        classes[r].style["top"] = "-"+String(taille)+ "px" ;
        classes[r].style["left"] = "-"+String(taille)+ "px" ;
      }
     }
    }
  });
```

styleSheets = la liste des feuilles de style associées à la page.
Classes = la liste des règles CSS d'une feuille de style.

La seconde approche peut être plus rapide dans le cas où il y a beaucoup de symbole, puisque l'on ne fait qu'une seule modification (celle de la définition du style) alors que dans la première approche on doit modifier tous les symboles un par un.

 Modifier votre couche de Grignon précédente, pour ajouter une étiquette (titre) qui devra disparaitre une fois atteint un certain niveau de zoom (plus petit que 10 par ex, 0 = le monde entier)

- couche.getTooltip() renvoie les étiquettes d'une couche si elles existent
- couche.unbindTooltip() enlève les étiquettes d'une couche
- couche.bindTooltip() ajoute des étiquettes

On peut jouer sur la propriété « permanent » (true ou false) pour faire apparaitre / disparaitre les étiquettes

 Faire afficher dans la console à l'aide d'une fonction associé à un évènement « **baselayerchange** »

```
function event_chg (e) {
    console.log(e.name); // The name of the layer
}

map.on('baselayerchange', event_chg);
```

■ *La digitalisation sur fond de plan (outils de dessin)*

 Modifier le script précédent pour que quand l'utilisateur clique cela ajoute un nouveau point dans une même couche geoJSON « inputLayer ».

- Le plugin leaflet.draw

Pour des fonctions plus avancées de digitalisation (dessin et édition), en particulier pour les polylignes et polygones, activer le plugin leaflet.draw,

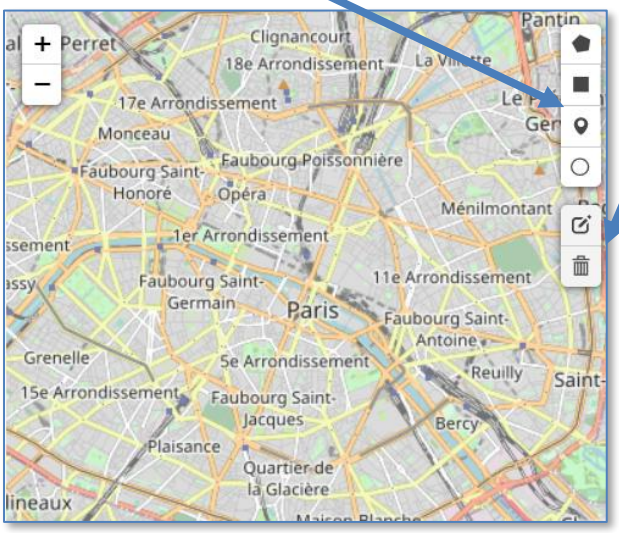
Dans la section <head> ajouter, en plus des directives habituelles leaflet, ce code :

```
<link rel="stylesheet" href="leaflet/leaflet.draw/src/leaflet.draw.css" />

<script src="leaflet/leaflet.draw/src/Leaflet.draw.js"></script>
<script src="leaflet/leaflet.draw/src/Leaflet.Draw.Event.js"></script>

<script src="leaflet/leaflet.draw/src/edit/handler/Edit.Poly.js"></script>
<script src="leaflet/leaflet.draw/src/edit/handler/Edit.SimpleShape.js"></script>
<script src="leaflet/leaflet.draw/src/edit/handler/Edit.Rectangle.js"></script>
<script src="leaflet/leaflet.draw/src/edit/handler/Edit.Marker.js"></script>
<script src="leaflet/leaflet.draw/src/edit/handler/Edit.CircleMarker.js"></script>
<script src="leaflet/leaflet.draw/src/edit/handler/Edit.Circle.js"></script>

<script src="leaflet/leaflet.draw/src/draw/handler/Draw.Feature.js"></script>
<script src="leaflet/leaflet.draw/src/draw/handler/Draw.SimpleShape.js"></script>
<script src="leaflet/leaflet.draw/src/draw/handler/Draw.Polyline.js"></script>
<script src="leaflet/leaflet.draw/src/draw/handler/Draw.Marker.js"></script>
<script src="leaflet/leaflet.draw/src/draw/handler/Draw.Circle.js"></script>
<script src="leaflet/leaflet.draw/src/draw/handler/Draw.CircleMarker.js"></script>
<script src="leaflet/leaflet.draw/src/draw/handler/Draw.Polygon.js"></script>
<script src="leaflet/leaflet.draw/src/draw/handler/Draw.Rectangle.js"></script>

<script src="leaflet/leaflet.draw/src/ext/TouchEvents.js"></script>
<script src="leaflet/leaflet.draw/src/ext/LatLngUtil.js"></script>
<script src="leaflet/leaflet.draw/src/ext/GeometryUtil.js"></script>
<script src="leaflet/leaflet.draw/src/ext/LineUtil.Intersect.js"></script>
<script src="leaflet/leaflet.draw/src/ext/Polyline.Intersect.js"></script>
<script src="leaflet/leaflet.draw/src/ext/Polygon.Intersect.js"></script>

<script src="leaflet/leaflet.draw/src/Control.Draw.js"></script>
<script src="leaflet/leaflet.draw/src/Tooltip.js"></script>
<script src="leaflet/leaflet.draw/src/Toolbar.js"></script>


<script src="leaflet/leaflet.draw/src/draw/DrawToolbar.js"></script>
<script src="leaflet/leaflet.draw/src/edit/EditToolbar.js"></script>
<script src="leaflet/leaflet.draw/src/edit/handler/EditToolbar.Edit.js"></script>
<script src="leaflet/leaflet.draw/src/edit/handler/EditToolbar.Delete.js"></script>
```

Puis dans le code de la page le code spécifique à leaflet.draw :

```
// création d'un groupe d'objets = regroupe les couches qui doivent être editables
 var drawnItems = new L.FeatureGroup();
 map.addLayer(drawnItems);

// ajout du contôle
 var drawControl = new L.Control.Draw({
        position: 'topright',
        draw: {
            polyline: false,
            polygon: true,
            circle: false,
            marker: true
        },
        edit: {
            featureGroup: drawnItems,
            remove: true
        }
     });
 map.addControl(drawControl);

 // gestion de l'évenement création
 map.on(L.Draw.Event.CREATED, function (e) {
  var type = e.layerType
  var layer = e.layer;

// Do whatever else you need to. (save to db, add to map etc)
  drawnItems.addLayer(layer);
});
// gestion de l'évenement édition
map.on(L.Draw.Event.EDITED, function (e) {
        var layers = e.layers;
        var countOfEditedLayers = 0;
        layers.eachLayer(function (layer) {
            countOfEditedLayers++;
        });
        console.log("Edited " + countOfEditedLayers + " layers");
    });
```

La barre d'outils de dessin et la barre d'édition



6. Gestion des projections dans Leaftlet

La projection par défaut dans Leaflet est la projection « EPSG:3857 » "Google Mercator" ("Web Mercator").

Certaines fonctions de lecture de données (GeoJSON par ex.) ne projettent pas directement les couches qui sont considérées comme étant en latitude / longitude.

Dans le dossier data le fichier znieff2.geojson, contient les ZNIEFF de niveau 2 sur la France en Lambert 93 (EPSG :2154)

Ajouter cette couche geojson, en utilisant la fonction readGeojson définie précédemment.

```
var znieffLayer = new L.GeoJSON(readGeojson("data/znieff2.geojson"));
znieffLayer.addTo(map);
```

Résultat : rien n'apparait

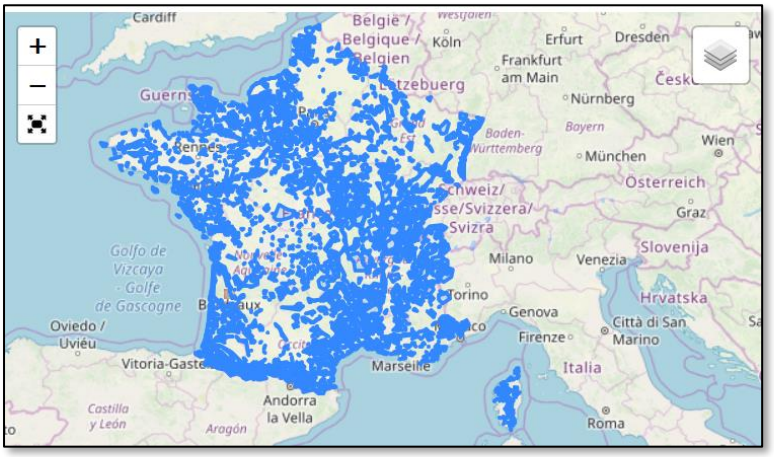
Ajoutez maintenant le plugin Proj4 de gestion des projections cartographiques :
A mettre juste après l'insertion de Leaflet en début de fichier.

```
  <script src="leaflet/proj4leaflet/lib/proj4.js"></script>
  <script src="leaflet/proj4leaflet/src/proj4leaflet.js"></script>
```

On va maintenant projeter la couche avec L.Proj.GeoJSON()

```
var crs = new L.Proj.CRS('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');

var znieffLayer = new L.Proj.GeoJSON(readGeojson("data/znieff2.geojson"));
znieffLayer.addTo(map);
```

Résultat : Les Znieff apparaissent



Cela fonctionne car si on regarde le début du fichier znieff2.geojson dans notepad++, on voit qu'il a bien son « crs » (2154) de défini :

```
{
"type": "FeatureCollection",
"crs": { "type": "name", "properties": { "name": "urn:ogc:def:crs:EPSG::2154" } },
"features": [
...
```

Dans le cas où un objet GeoJSON (oJSON) n'aurait pas son CRS de défini, on peut le définir « à la main » (il faut le connaitre) par :

```
oJSON.crs =  {
  "type": "name",
  "properties": {
    "name": "EPSG:2154"
    }
  }
```

Le code ci-dessous sert à définir le Lambert 93 qui n'est pas par défaut dans la librairie leaflet / proj4 :

```
var crs = new L.Proj.CRS('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');
```

Pour ajouter une nouvelle projection :

Aller sur le site : http://spatialreference.org/ref/epsg/
pour récupérer la définition de la projection (le code EPSG est un code international des projections)

2154 est le Lambert 93
copier / coller la définition en proj4js (javascript) dans votre code javascript
Proj4js.defs["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";

Puis modifier comme dans le code de l'encart ci-dessus.
EPSG des principales projections :

RGF93 Lambert 93 (EPSG = 2154)
Projections "Conique Conforme 9 zones" Lambert 93 CC (CC42 à CC50) EPSG de 3942 à 3950

NTF Lambert Zone 1 "Nord"	27561
NTF Lambert Zone 2 "Centre"	27562
NTF Lambert Zone 3 "Sud"	27563
NTF Lambert Zone 4 "Corse"	27564
NTF Lambert Zone 1 "Carto"	27571
NTF Lambert Zone 2 "Carto" et Lambert II étendu	27572
NTF Lambert Zone 3 "Carto"	27573
NTF Lambert Zone 4 "Carto"	27574

UTM 30 Nord (EPSG = 32630)
UTM 31 Nord (EPSG = 32631)
UTM 32 Nord (EPSG = 32632)

7. Gécodage et adresse

Il s'agit de trouver une position correspondent à une adresse, ou inversement des informations (commune, département …) correspondent à une position.

■ *Le plugin leaflet-control-geocoder*

Insérer le plugin :

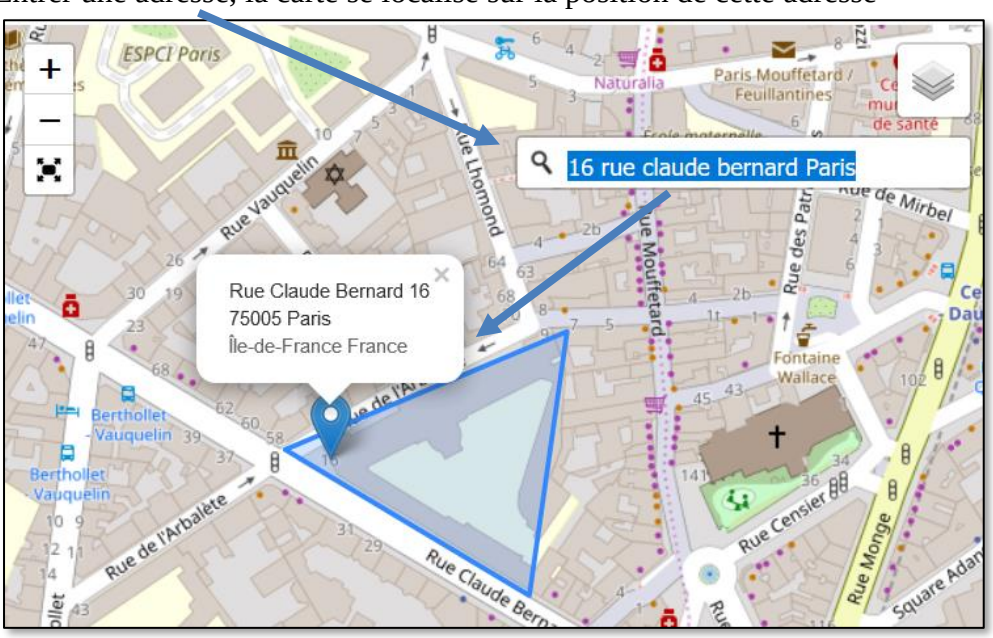
```
<link rel="stylesheet" href="leaflet/leaflet-control-
geocoder/dist/Control.Geocoder.css" />
<script src="leaflet/leaflet-control-geocoder/dist/Control.Geocoder.js"></script>
```

Ajouter le contrôle du plugin :

```
L.Control.geocoder().addTo(map);
```

La loupe de recherche est ajoutée

Entrer une adresse, la carte se localise sur la position de cette adresse



■ *Utilisation des API IGN : trouver le nom de commune et de département d'une position:*

Modifier votre fonction GoTo pour qu'elle affiche en bas de page le nom de la commune et du département qui correspondent à la position latitude, longitude donnée par l'utilisateur.

On va créer deux fonctions javascript GetCommune() et GetDepartement() pour cela :

```
function GetCommune(lat,lo){

var xmlhttp = new XMLHttpRequest();

lien="https://geo.api.gouv.fr/communes?fields=code,nom&lat=";
lien += String(lat) + "&lon=" + String(lo);

xmlhttp.open("GET", lien, false);
xmlhttp.overrideMimeType("application/json");
xmlhttp.send();
objGEO=JSON.parse(xmlhttp.responseText);

return objGEO[0].nom;

}

function GetDepartement(lat,lo){

var xmlhttp = new XMLHttpRequest();

lien="https://geo.api.gouv.fr/communes?fields=departement&lat=";
lien += String(lat) + "&lon=" + String(lo);

xmlhttp.open("GET", lien, false);
xmlhttp.overrideMimeType("application/json");
xmlhttp.send();
objGEO=JSON.parse(xmlhttp.responseText);

return objGEO[0].departement.nom;

}
```

Le lien https://geo.api.gouv.fr est une API qui permet de demander ces informations au serveur IGN, le résultat est demandé en format JSON
Dans le navigateur entrer l'adresse « à la main » pour Amiens

https://geo.api.gouv.fr/communes?fields=departement&lat=49.89&lon=2.29

Le Résultat est un objet en format JSON :

Le format JSON = JavaScript Object Notation

JavaScript Object Notation (JSON) est un format de données textuelles dérivé de la notation des objets du langage JavaScript. Il permet de représenter de l'information structurée comme le permet XML par exemple. Il existe une variante pour les données géographiques le GeoJSON.

Ajouter sous le bouton GoTo un texte où va être écrite l'information : **<h3 id="info">info</h3>**
Le champ 'id' donne un identifiant pour notre texte

```
com = GetCommune(lat,lon);
dep = GetDepartement(lat,lon);
msg = "département= ".concat(dep," commune= ",com);
document.getElementById('info').innerHTML = msg;

}
```

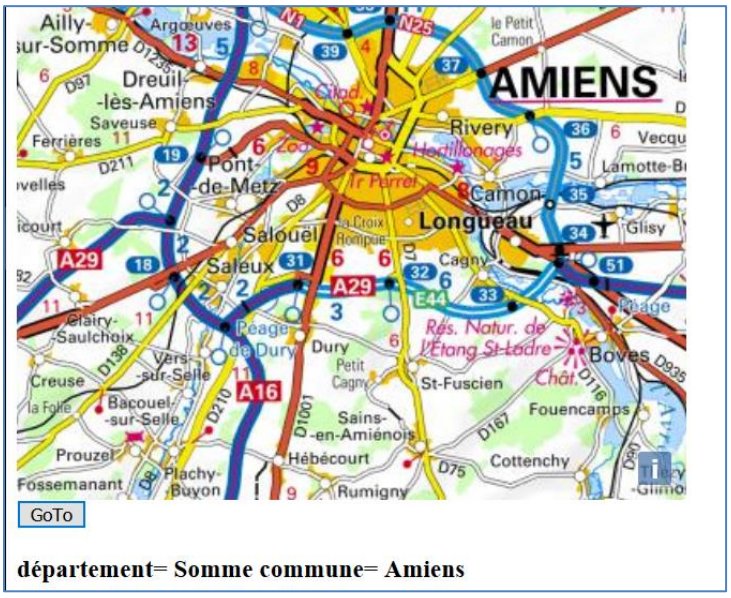
La fonction .concat() permet de concaténer des chaines de caractères en une seule.

document = objet javascript contenant toute la page

.getElementById(« info ») permet de manipuler un objet identifier de la page, ici notre texte.

.innerHTML permet de modifier l'objet de la page

Le résultat sur la position de la ville d'amiens:



■ *L'API pour faire le géocodage en javascript:* https://api-adresse.data.gouv.fr

Pour trouver la position de l'adresse 16 rue claude bernard à Paris :

https://api-adresse.data.gouv.fr/search/?q=16+rue+claude+bernard+paris

Le retour est un objet JSON avec les coordonnées de l'adresse :

JSON Données brutes En-têtes

Enregistrer Copier

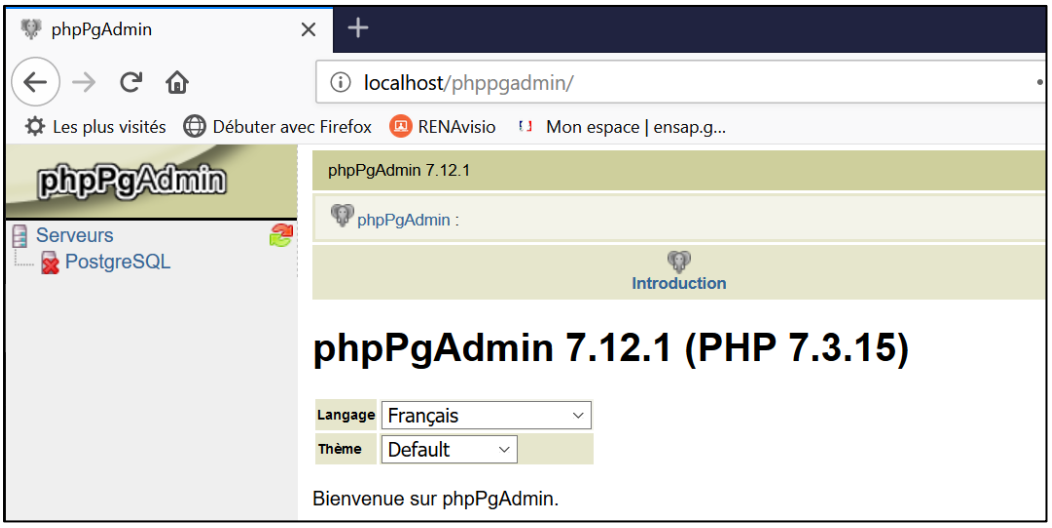

```
licence:                "ODbL 1.0"
limit:                  5
features:
  0:
    geometry:
      coordinates:
          0:            2.348786
          1:            48.839317
        type:           "Point"
      type:             "Feature"
    properties:
        context:        "75, Paris, Île-de-France"
        y:              6860115.6
        citycode:       "75105"
        postcode:       "75005"
        score:          0.946481818181818
        street:         "Rue Claude Bernard"
        city:           "Paris"
```


On peut ainsi géocoder automatiquement tout un fichier d'adresses, avec un script javascript.

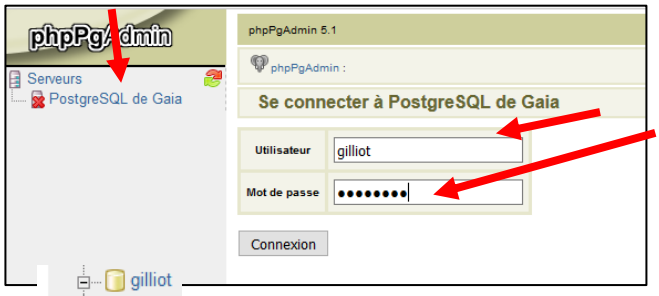
III. Gestion des données géographiques en Base de données : PostgreSQL+ PostGIS

1. Utilisation de l'outil PhpPgAdmin pour gérer la base de données PostgreSQL

Ouvrir dans votre navigateur internet le site : http://localhost/pgadmin/



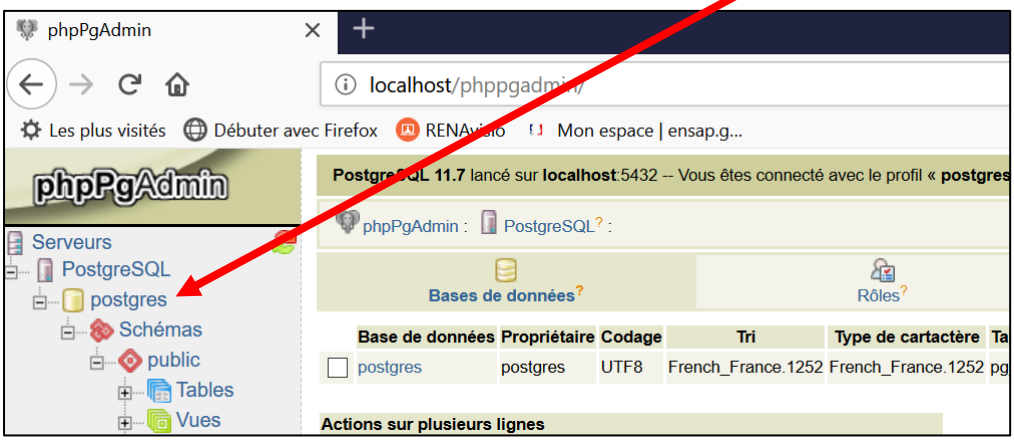
Cliquez sur PostgreSQL



Rentrez votre identifiant (postgres / postgres)
Connectez-vous

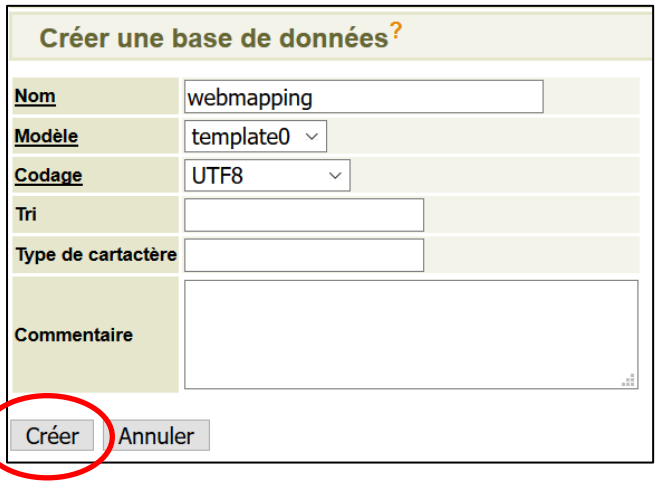
⚠ Sur macOS (votre_login / pas de mot de passe)

Il n'y a pour le moment qu'une seule base de données (postgres) qui est la base système.

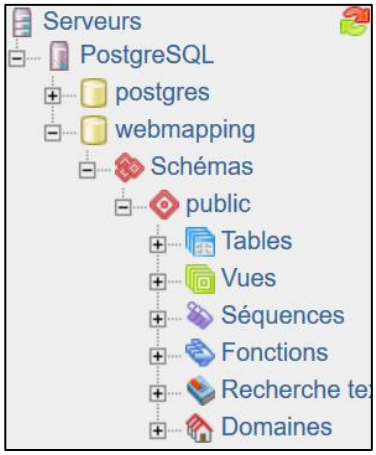


● Créer une base de données « **webmapping** » :

Cliquez sur PostgreSQL puis Créer une base de données



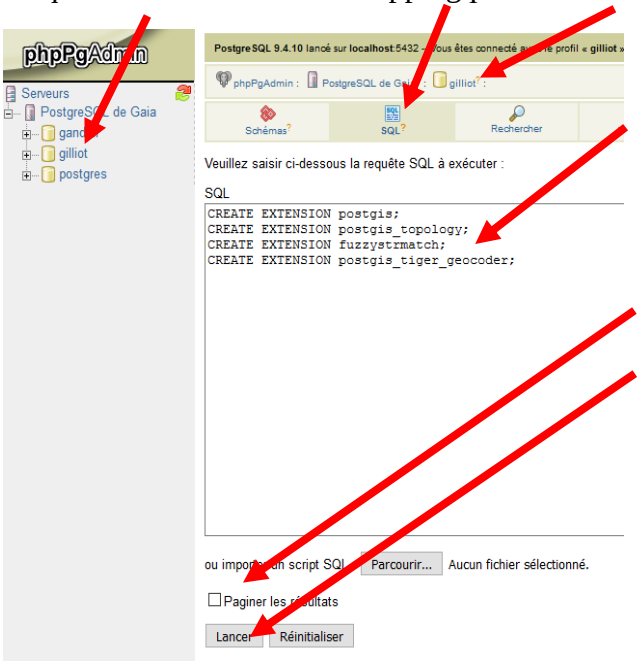
Votre nouvelle base « webmapping » apparait dans la liste des bases



Ajout des informations PostGIS dans votre base :

PostGIS est une extension facultative de postgreSQL, il faut l'activer :

Cliquez sur votre base webmapping puis dans l'onglet SQL bien vérifier que vous êtes dans votre base



Entrez le code suivant

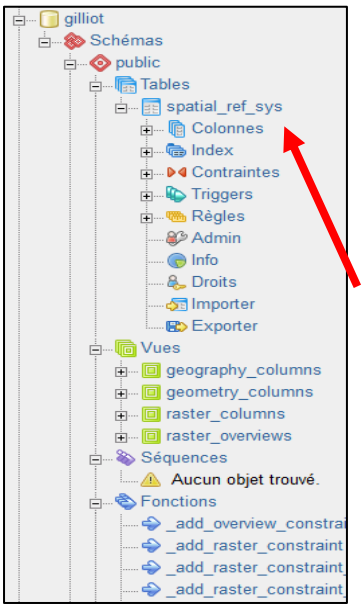
CREATE EXTENSION postgis;
CREATE EXTENSION postgis_topology;
CREATE EXTENSION fuzzystrmatch;
CREATE EXTENSION postgis_tiger_geocoder;

Décochez Paginer les résultats

Enfin Lancer l'exécution des commandes

Regardez maintenant le contenu de votre base de données : webmapping

Une table a été créée : spatial_ref_sys

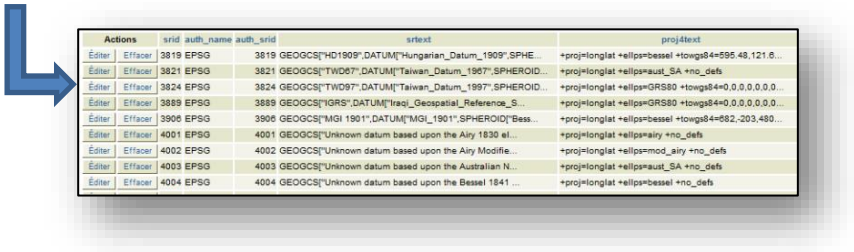


Des vues ont été ajoutées

Ainsi que de nombreuses fonctions

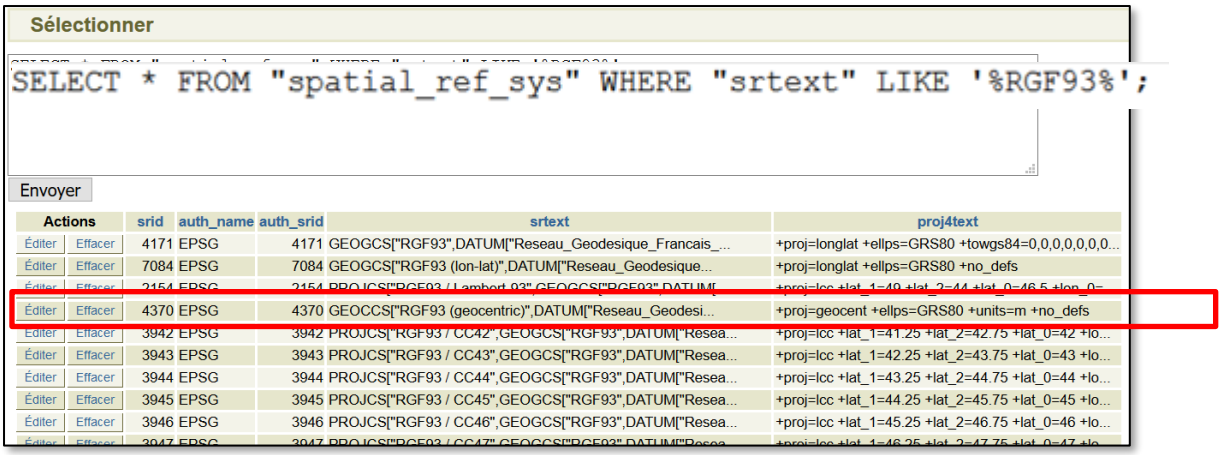
La table spatial_ref_sys contient les définitions des référentiels géographiques.

Afficher la table : 1) Cliquez sur la table 2) Parcourir



Par une requête SQL (champ srtext) affichez les référentiels français RGF93.

*SELECT * FROM "spatial_ref_sys" where "srtext" like '%RGF93%';*



Attention à l'usage des guillemets pour le nom des champs et des apostrophes pour les constantes chaines de caractères.

Notez le référentiel de srid 2154 (RGF93 Lambert 93) c'est le référentiel « officiel » français, le plus utilisé.
Notez l'opérateur SQL LIKE combiné au caractère % , pour retrouver tous les systèmes qui contiennent la chaine de caractères 'RGF93'.

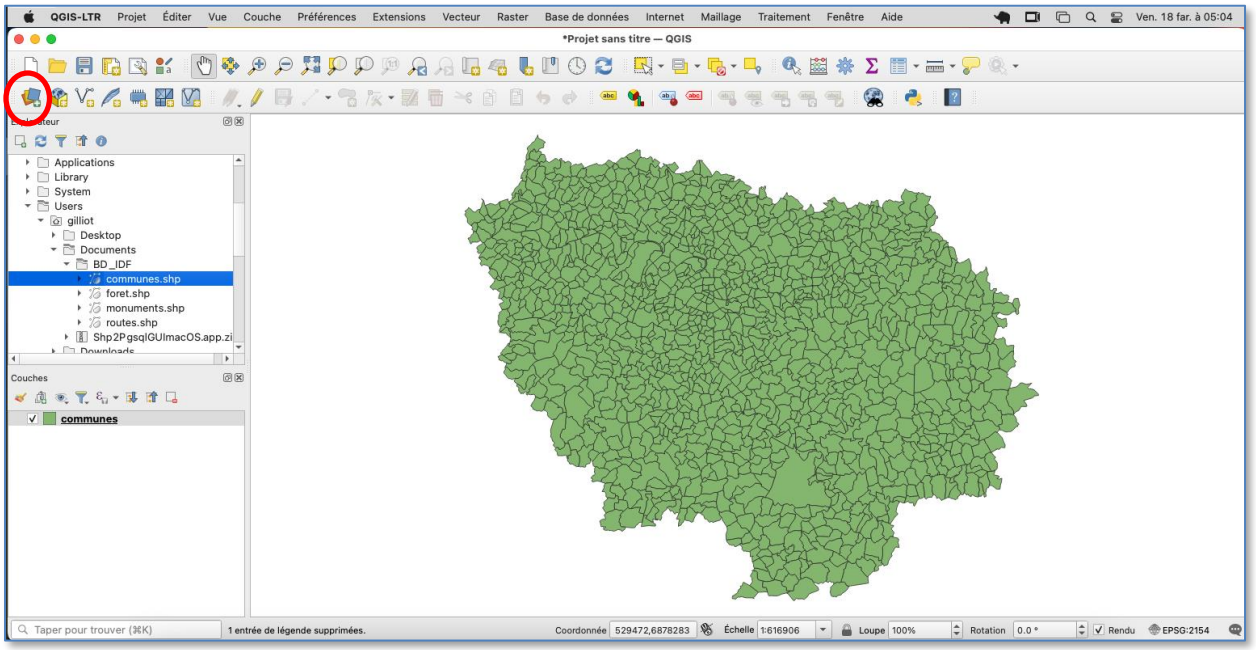
2. Importation de données géographiques dans POSTGIS

2.1. Importation des shapefiles avec le logiciel QGIS

QGIS est un logiciel généraliste de SIG (système d'informations géographiques), il peut être utile pour préparer les données à mettre dans la base de données POSTGIS et il va nous permettre de facilement importer de nouvelles données dans notre base.

Lancez QGIS, on va importer dans la base la couche des communes d'Ile de France.

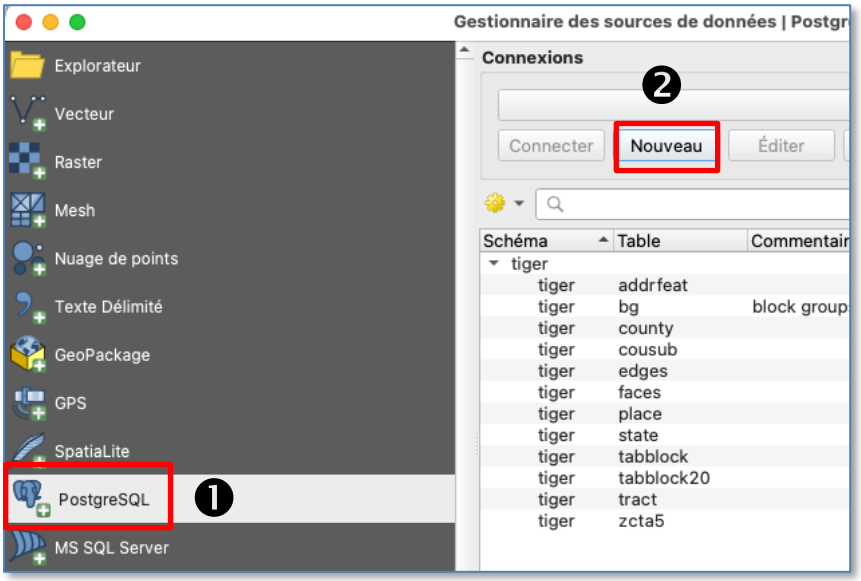
Dans le panneau explorateur à gauche allez dans vos documents dans le dossier « BD_IDF »
Cliquer ou faire glisser « communes.shp » vers la carte



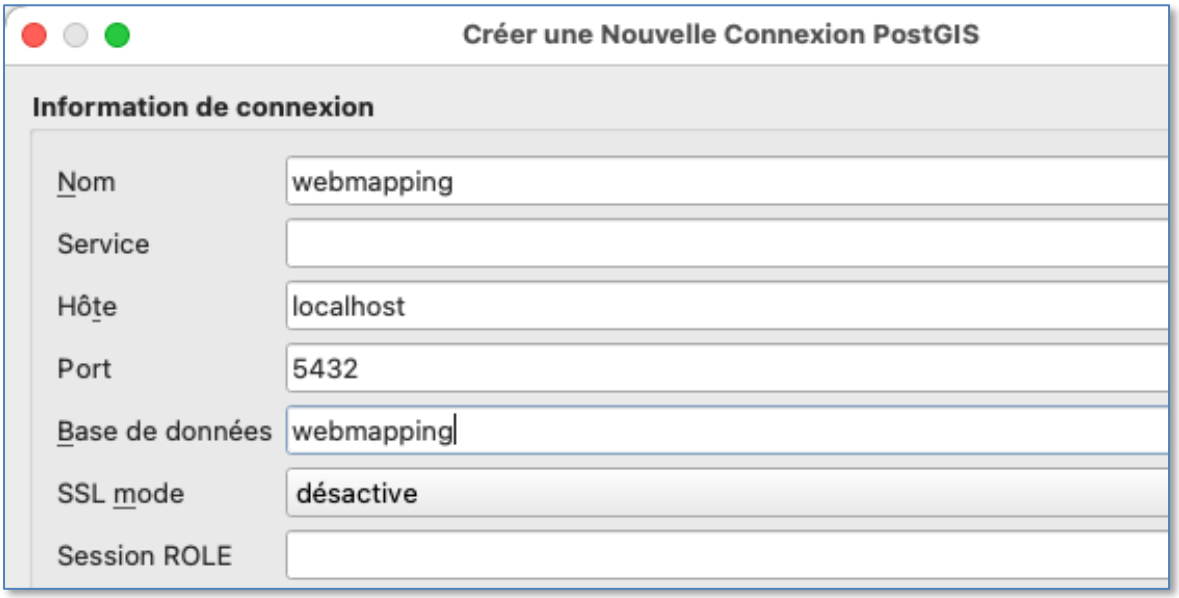
Connexion à la base de données POSTGIS

Cliquez sur le bouton  en haut à gauche

Choisir à gauche le type de base PostgreSQL et cliquer sur nouveau

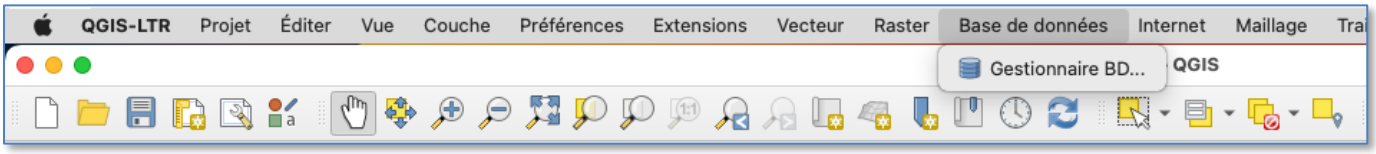


Rentrez les paramètres de connexion à notre base de données « webmapping » :



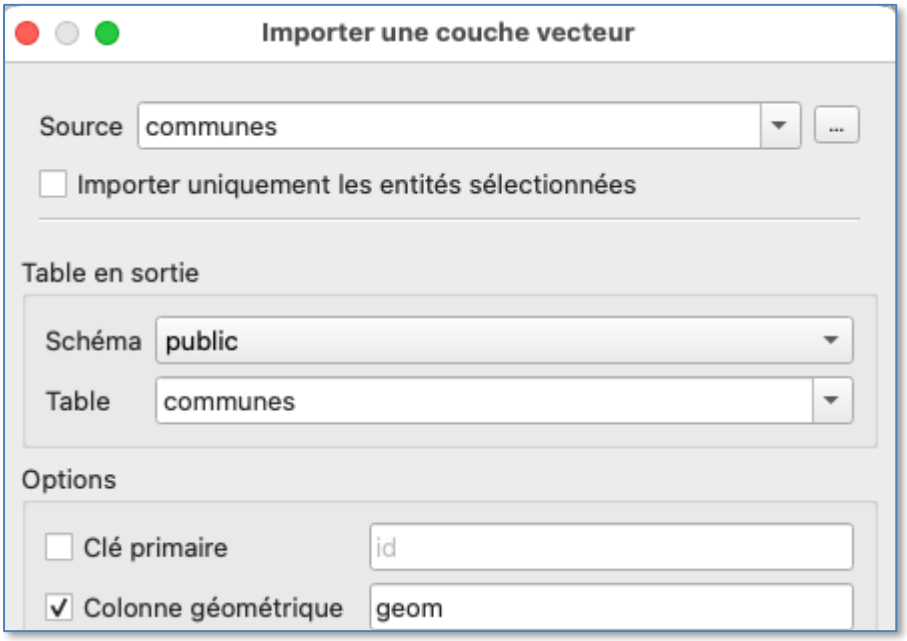
Valider par le bouton OK
Se connecter avec le bouton Connecter

Ouvrir le Gestionnaire de Base de données depuis le Menu QGIS « Base de données »

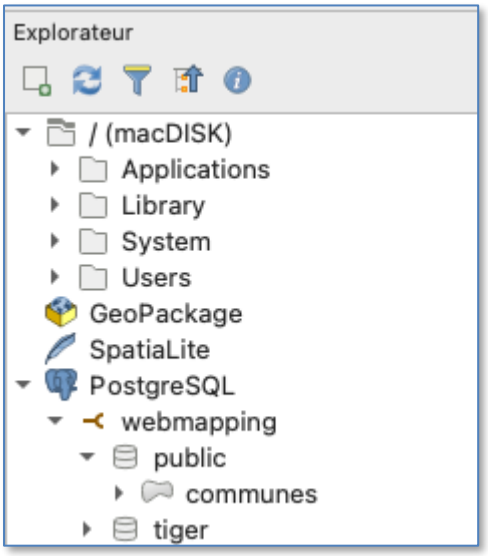


1) Cliquer sur notre base de données Webmapping
2) Puis cliquer sur le bouton Import de couche/Fichier

Choisir comme Source la couche « communes » qui a été chargée dans QGIS et cocher colonne géographique



Regarder en bas du volet « Explorateur » de QGIS

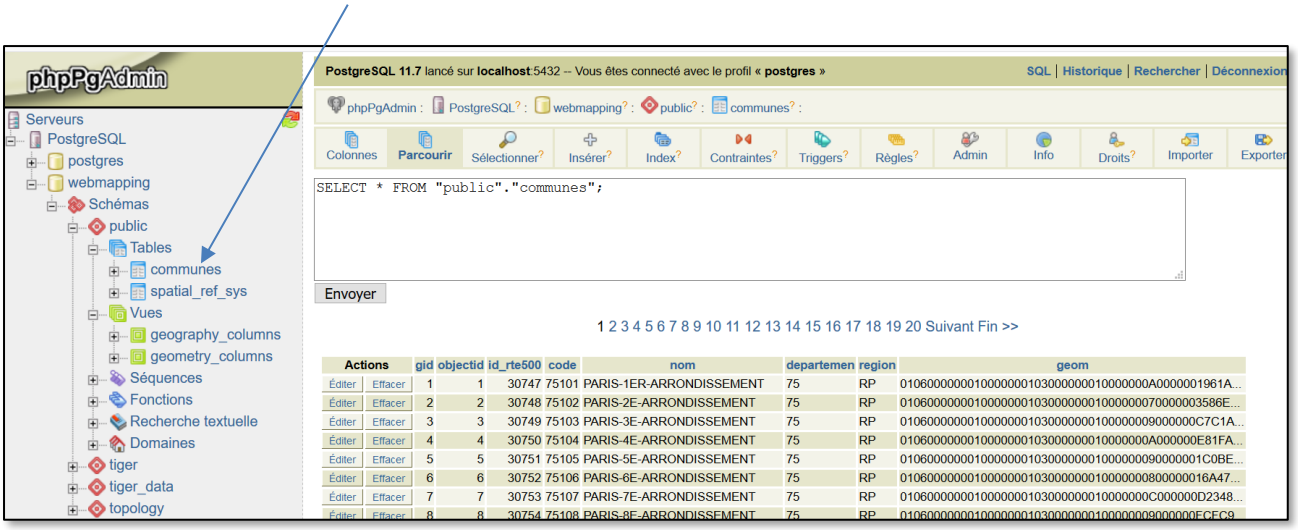


La connexion PostgreSQL est visible

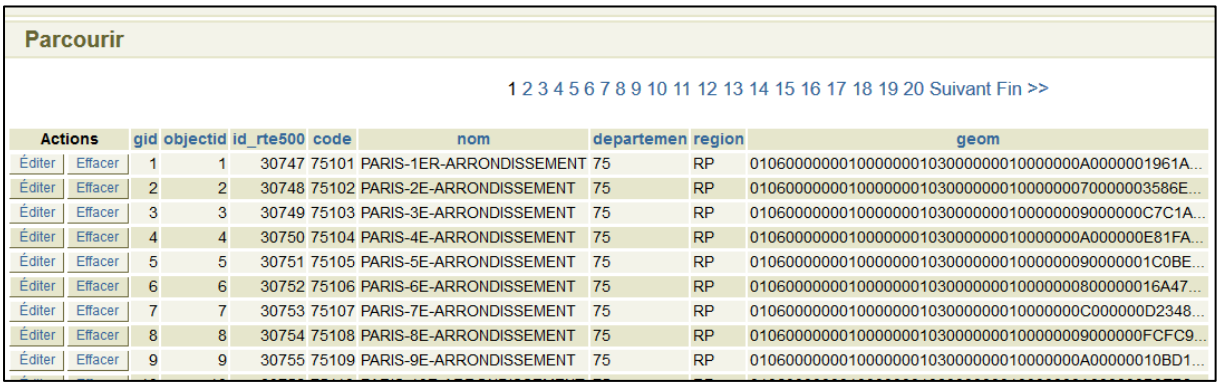
La table « communes » a été importée

Retournez maintenant dans phppgadmin et rafraichir la page sous le navigateur internet :

Une nouvelle table « communes » apparait dans votre base :

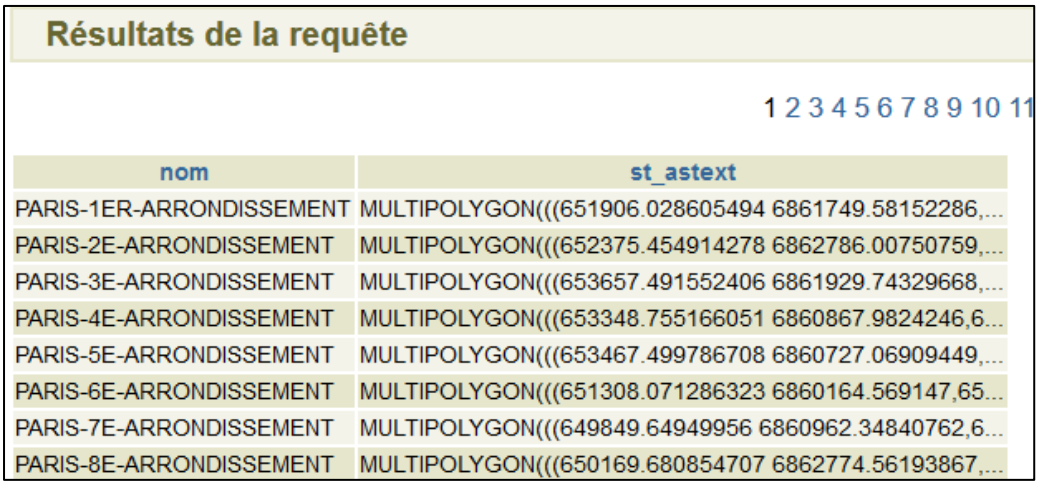


Affichez la table « communes » et notez le champ « ***geom*** » de cette table qui contient les polygones des communes



On peut afficher les données du champ « geom » de manière lisible par une requête SQL :

Select nom, ST_AsText(geom) from communes



Vérifier le référentiel géographique (SRID)

```
SELECT Find_SRID('public','communes', 'geom');
```

Fixer le référentiel géographique en Lambert 93 (SRID=2154) si non défini (0)

```
ALTER TABLE communes
  ALTER COLUMN geom
    TYPE geometry(MultiPolygon, 2154)
    USING ST_SetSRID(geom, 2154);
```

De la même façon importer dans la base **monuments.shp**, **routes.shp** et **foret.shp**

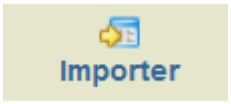
Importer des données depuis un fichier CSV de type X,Y (avec des coordonnées géographiques)

Le fichier **station78.csv** contient les stations de traitement d'eau des Yvelines, avec les champs lat et lon pour la latitude / longitude.

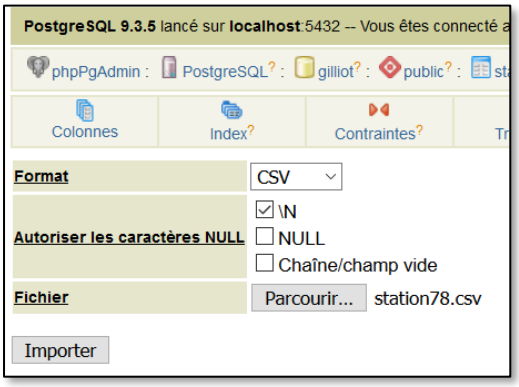
	A	B	C	D	E	F
1	nom	lat	lon	cd_commune	commune	capacite
2	ABLIS LES VIG	48.5249327	1.8235055	78003	ABLIS	13500
3	ABLIS MAING	48.5376449	1.8445981	78003	ABLIS	200
4	AUBERGENVI	48.981879	1.8429353	78029	AUBERGENVI	17200
5	AUFFARGIS-B	48.7011147	1.8952617	78030	AUFFARGIS	2000
6	BAZAINVILLE	48.7949823	1.6568871	78048	BAZAINVILLE	1500

Créer la structure de la table qui va recevoir ces données :

```
CREATE TABLE station78(
  nom character varying,
  lat real,
  lon real,
  cd_commune integer,
  commune character varying,
  capacite integer
);
```



dans la table que vous venez de créer le fichier CSV



La table résultat dans PostgreSQL :

nom	lat	lon	cd_commune	commune	capacite
ABLIS LES VIGNES	48.5249	1.82351	78003	ABLIS	13500
ABLIS MAINGUERIN	48.5376	1.8446	78003	ABLIS	200
AUBERGENVILLE	48.9819	1.84294	78029	AUBERGENVILLE	17200
AUFFARGIS-Bourg	48.7011	1.89526	78030	AUFFARGIS	2000
BAZAINVILLE	48.795	1.65689	78048	BAZAINVILLE	1500

Pour le moment c'est une table « normale », il faut créer une géométrie pour qu'elle soit une table géographique : on ajoute une colonne de type GEOMETRY (type point), 4326 est le référentiel géographique pour latitude / longitude (WGS84°), puis on met à jour (UPDATE) cette géométrie à partir des champs lon et lat.

```
ALTER TABLE station78 ADD COLUMN geom GEOMETRY(POINT,4326);
UPDATE station78 SET geom = ST_SetSRID(ST_POINT(lon,lat),4326);
```

```
SELECT ST_AsText(geom) FROM station78;
```

st_astext
POINT(1.82350552082062 48.5249328613281)
POINT(1.84459805488586 48.5376434326172)

2.2. Importation des shapefiles avec le logiciel Shp2pgsql-GUI = autre méthode (ANNEXE 1)
2.3. Importation par la ligne de commande = autre méthode (ANNEXE 1)

3. Accès aux tables géographiques de POSTGIS depuis Leaflet

Comme la base de données est « côté serveur » et Leaflet est « côté client » (dans le navigateur internet), Il faut sur le serveur un script en langage PHP qui extrait de la base de données les informations qui nous intéresse et les envoie au navigateur.

C'est le script **sql_postgis.php** à la racine du serveur (dossier du site web).

Le fichier est à la racine du dossier htdocs

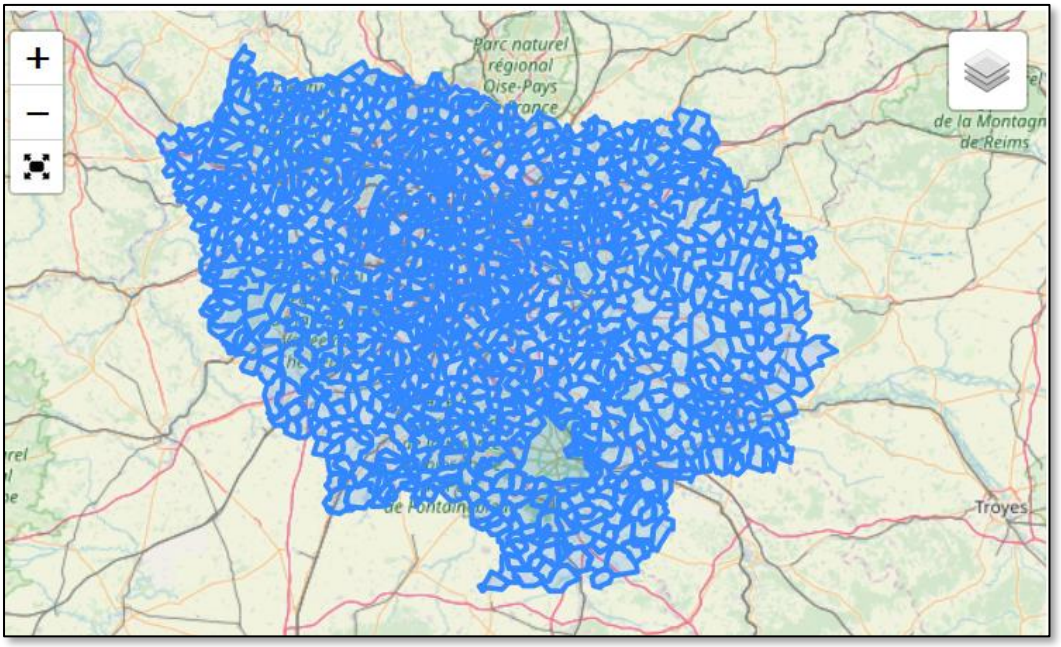
Ce script va être appelé par javascript en lui passant en argument une requete SQL ($sql = $_GET["sql"];) et il renvoie les données en format geoJSON.

```php
<?php

$sql = $_GET["sql"];

$conn = pg_connect("dbname='webmapping' user='postgres' password='postgres'
host='localhost' port=5433");
if ($conn) {
    # Build GeoJSON feature collection array
$geojson = array(
   'type'      => 'FeatureCollection',
   'features'  => array()
);

$req = pg_query($sql);

# Loop through rows to build feature arrays
while ($row = pg_fetch_assoc($req)) {
    $properties = $row;
    # Remove geojson and geometry fields from properties
    unset($properties['geom']);
    unset($properties['geojson']);
    $feature = array(
         'type' => 'Feature',
         'geometry' => json_decode($row['geojson'], true),
         'properties' => $properties
    );
    # Add feature arrays to feature collection array
    array_push($geojson['features'], $feature);
}

header('Content-type: application/json');
echo json_encode($geojson, JSON_NUMERIC_CHECK);

pg_close();
}
else echo("erreur de connection");
?>
```

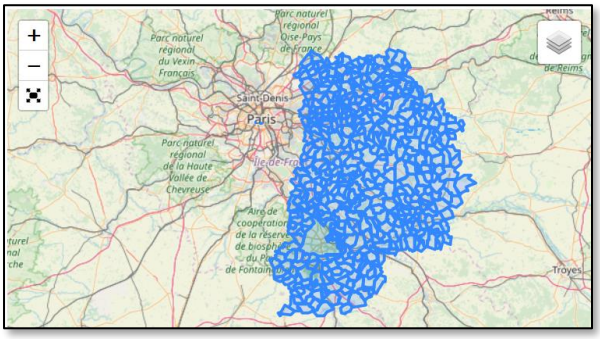
Dans la page javascript on ajoute le code suivant :

```
function sql_postgis(sql){
    var xmlhttp = new XMLHttpRequest();
    xmlhttp.open("GET", "sql_postgis.php?sql=" + sql, false);
    xmlhttp.send();
    return JSON.parse(xmlhttp.responseText);
}

var sql="SELECT *, ST_AsGeoJSON(ST_Transform(geom,4326)) as geojson from communes";

oJSON = sql_postgis(sql);

var geojsonCOM = new L.GeoJSON(oJSON);

geojsonCOM.addTo(map);
```

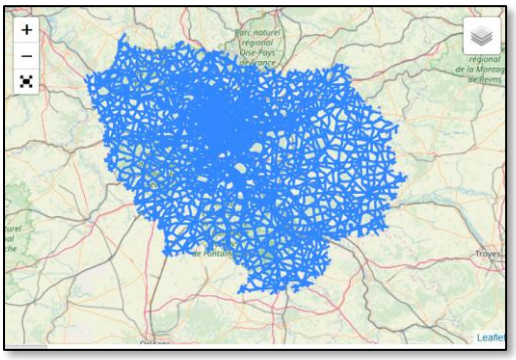
Le résultat : les communes d'Ile de France depuis la Base de données POSTGIS
Noter la fonction PostGIS ST_ASGeoJSON() qui convertie la géométrie en format GeoJSON et ST_Transform qui projette la couche en WGS84 (4326) pour Leaflet.
Le résultat :



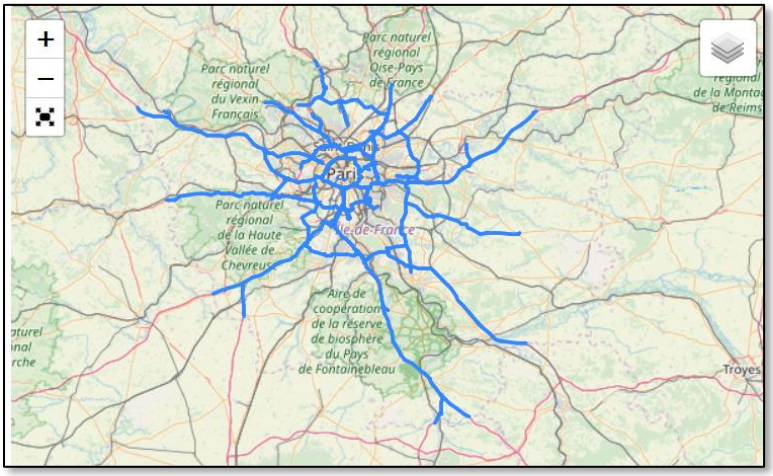
Modifier la requête afin de ne récupérer que les communes du département de Seine et Marne (77), le champ « departemen » contient le code de département.



Afficher les routes de la région depuis la couche « routes »



Depuis les routes précédentes, n'afficher que les autoroutes. Le champ « vocation » des routes contient 'Type autoroutier' pour les autoroutes.



 Créer une nouvelle couche « autoroute » dans la base de données et la cartographier.

```
sql_postgis("DROP TABLE IF EXISTS autoroutes");
sql_postgis("CREATE TABLE autoroutes AS SELECT * FROM routes WHERE vocation='Type autoroutier'");
```

DROP TABLE supprime une table IF EXISTS vérifie qu'elle existe pour éviter une erreur.
CREATE TABLE … AS va créer une table d'après le résultat d'une requête (SELECT …)

 Afficher depuis la base POSTGIS, une couche des stations de traitement de l'eau des Yvelines (table station78) avec un symbole circulaire dont la taille est proportionnelle à la capacité de la station (champ capacite en équivalent habitant)

4. Les fonctions géographiques de PostGIS

Dans phpPgAdmin

■ *Calculer des caractéristiques géométriques*

ST_Area(geom) permet de calculer la surface de chaque polygone.

 Afficher par une requête SQL le nom et la surface des communes **en ha**, donner le nom « surface » au champ calculé :

PARIS-20E-ARRONDISSEMENT	596.330095844902
SAINT-MANDE	1641.8241407143
ACHERES-LA-FORET	1251.06690832172
AVON	390.661896332698
AMILLIS	2018.8757275163
AMPONVILLE	1194.31903154425

 Ajoutez le calcul du périmètre en KM à l'aide de la fonction ST_Perimeter

SAINT-MANDE	22.6142494920462
ACHERES-LA-FORET	15.0773676190618
AVON	10.3870307116887
AMILLIS	20.6777870852972
AMPONVILLE	15.8912313994141
ANDREZEL	15.4232776533452

 Afficher par une requête SQL la surface totale des communes par département en ha

departemen	surface
93	23694.8699650796
75	8901.48439757893
92	17553.0126504153
95	125262.627989218
78	230634.197056943
91	181845.786437426
94	26102.2325585641
77	592389.305677712

La fonction SQL sum() calcule la somme d'un champ entre les lignes
La directive SQL group by « col » regroupe des lignes ensemble selon le champ « col »
Le mot clé as permet de renommer une colonne ou un calcul

■ *Requête spatiale*

Par une requête spatiale Afficher tous les monuments qui sont dans les communes du département des Yvelines. L'information département n'est pas présente dans la table de données des monuments : c'est pour cela qu'il faut une requête spatiale.

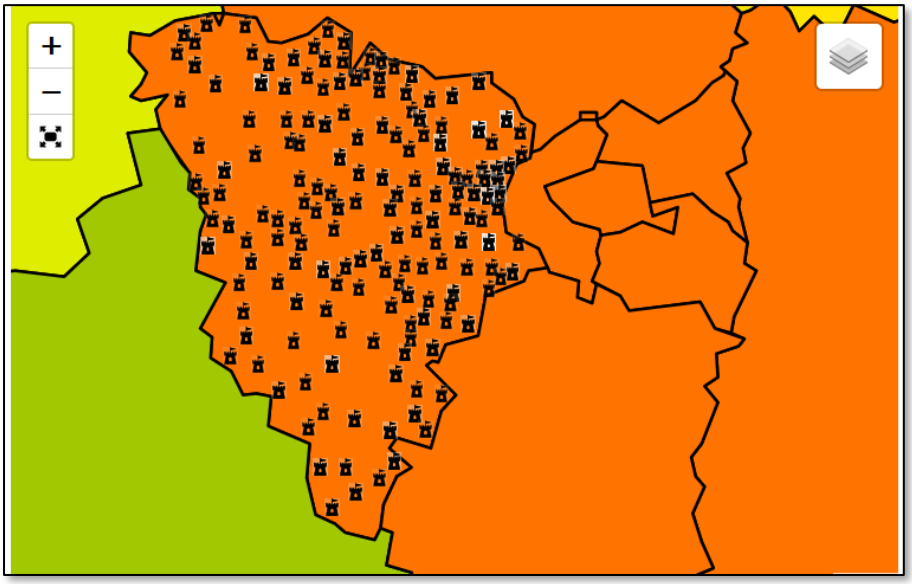
La function ***ST_Within(gA ,gB)*** retourne VRAI si gA est dans gB
gA et gB sont des géométries PostGIS (le champ geom)
ST_Contains(gA, gB) est la relation réciproque gA contient gB

```
select monuments.* from monuments, communes WHERE
ST_Within(monuments.geom,communes.geom) and communes.departemen='78'
```

Après intégration dans Leaflet :

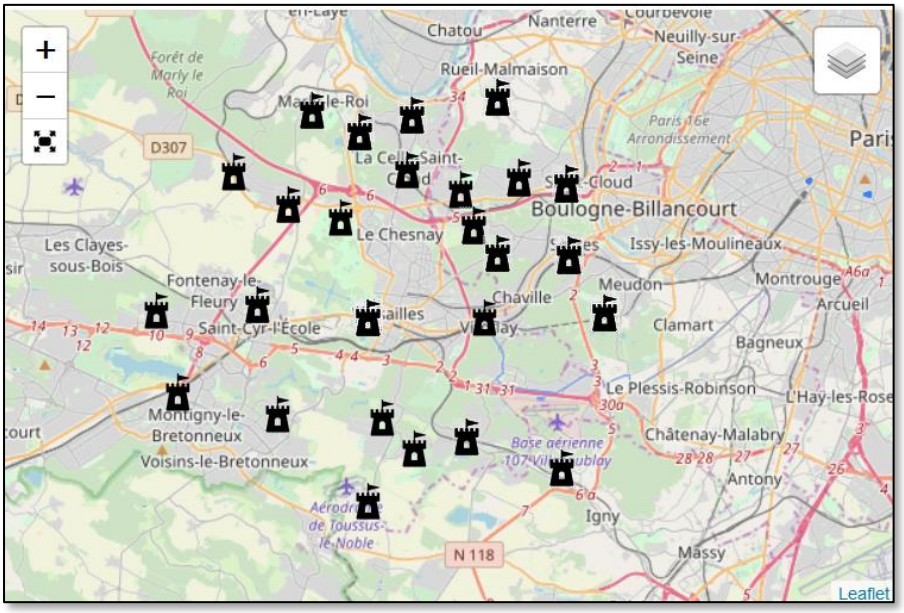
```
var sql="SELECT monuments.*, ST_AsGeoJSON(monuments.geom) as geojson from
monuments,communes where ST_Within(monuments.geom, communes.geom) AND
communes.departemen='78'";
var oJSON = sql_postgis(sql);


function fcIcon(feature,latlng){
 var mIcon = L.icon({ iconUrl: "data/chateau.png", iconSize: [10,12] });
 return L.marker(latlng, { icon: mIcon });
}

var geojsonCOM = new L.GeoJSON(oJSON,{pointToLayer:fcIcon});

geojsonCOM.addTo(map);
```

Le résultat :



Par une requête spatiale Afficher tous les monuments qui sont à moins de 5 km de la commune de VERSAILLES.

ST_Distance(gA, gB) calcule la *plus courte* distance entre deux géométries

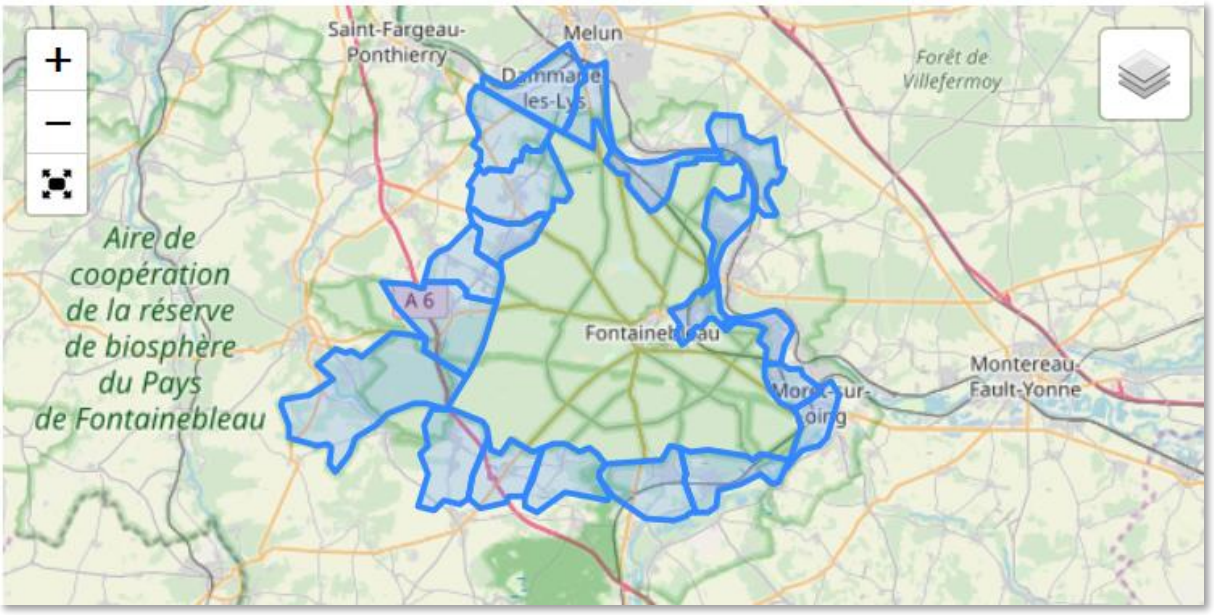


Par une requête spatiale Afficher tous les monuments qui sont à moins de 1 km d'une autoroute.

Par une requête spatiale Afficher toutes les communes limitrophes de la commune de FONTAINEBLEAU

ST_Touches teste si deux géométries se touchent en leur contours extérieurs

Dans une réquête on peut faire apparaitre deux fois la même table avec un alias différent : SELECT a.* FROM table as a, table as b WHERE …



D'autres relations spatiales sont utilisables : Voir documentations de postGIS.

■ *Jointure spatiale*

Par une jointure spatiale avec les communes « importez » dans les Châteaux (depuis la couche des monuments) le code du département dans lequel il est. Mettez une étiquette avec ce code. Dans la couche monuments le **champ tico** contient le type de monument, il commence par « Château » pour les châteaux.

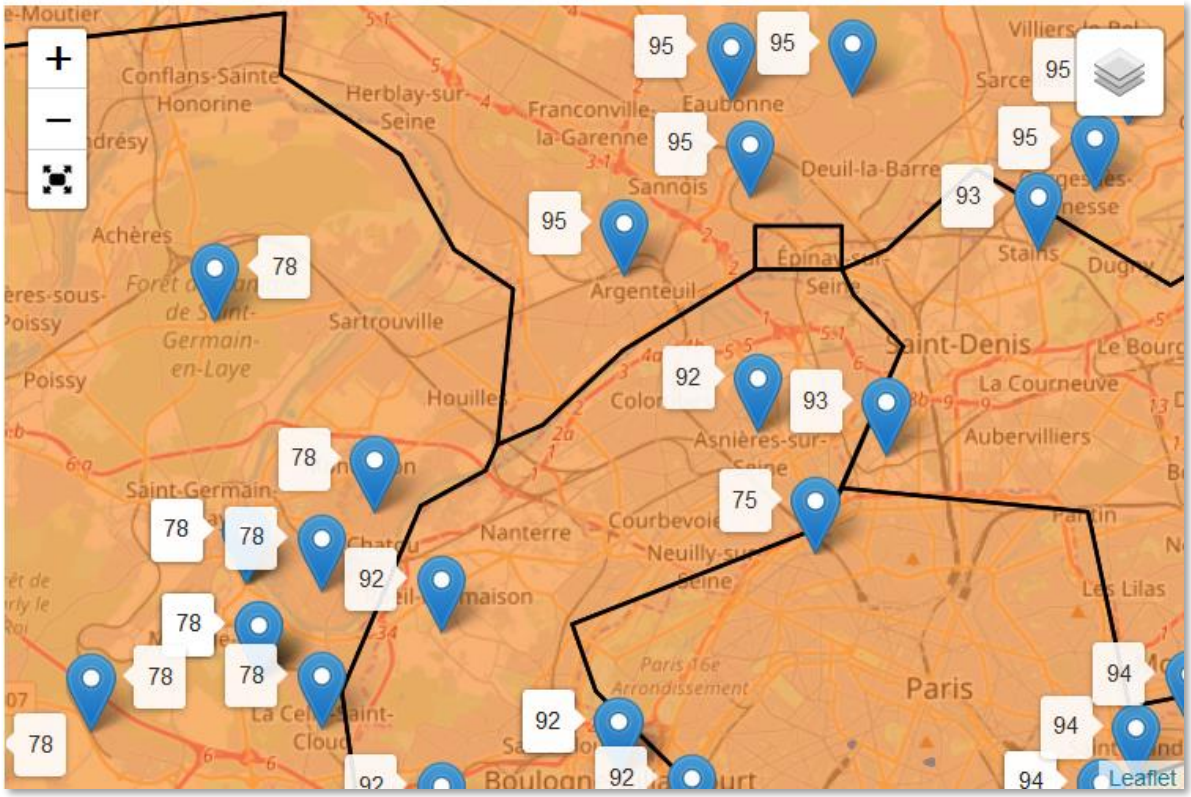
Pour une jointure spatiale le critère de jointure entre 2 couches A et B se présente sous la forme suivante :

SELECT … FROM … JOIN …
ON **ST_Within(A.geom, B.geom)**

La relation géographique est "A est inclus dans B"

La fonction SQL LEFT(chaine, n) renvoie les n premiers caractères en partant de la gauche d'une chaine de caractères.

La fonction javascript String() permet de convertir une valeur en une chaine de caractères.



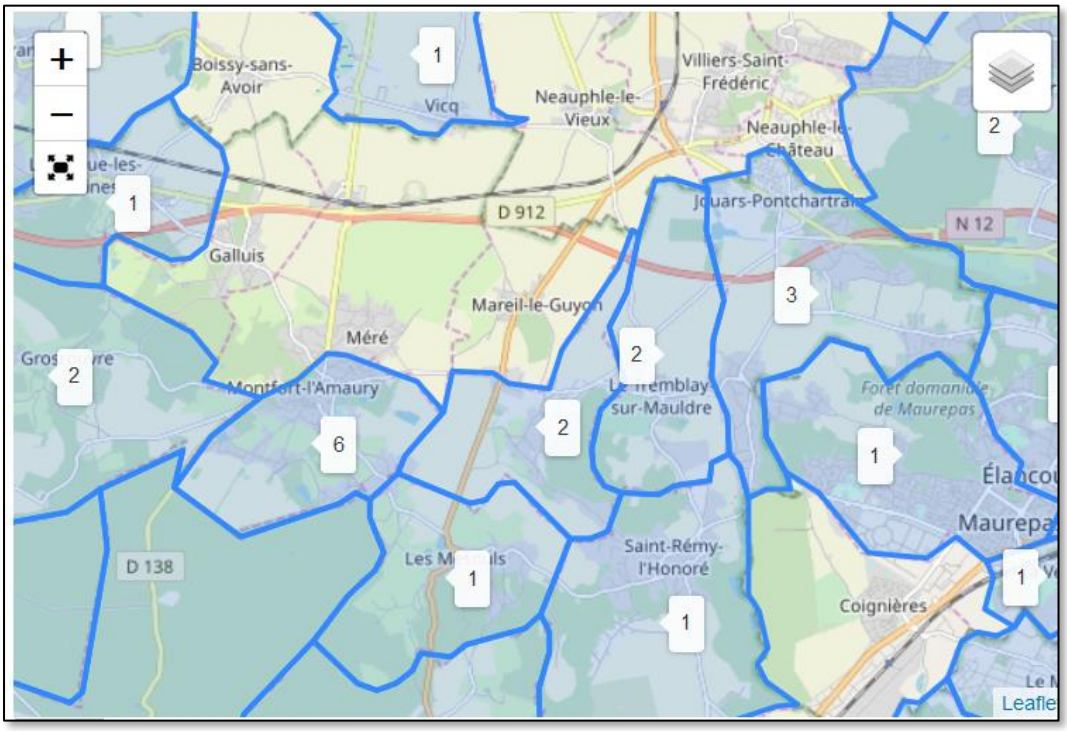
D'autres relations existent comme pour les requêtes spatiales (voir documentation).

■ *Regroupement*

Compter le nombre de monuments par commune et mettre une étiquette par commune avec ce nombre.

```
SELECT C.*, T.nb, ST_AsGeoJSON(C.geom) as geojson FROM communes as C
JOIN (SELECT insee, count(*) as nb FROM monuments GROUP BY insee) T
ON C.code = T.insee
```

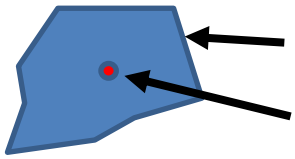
code est le code INSEE dans la couche commune
insee est ce même code INSEE dans les monuments
C et T sont des alias pour alléger la syntaxe (abréger le nom des tables)
nb est le champ calculé qui donne le nombre de monuments.
Remarquez l'utilisation d'une **sous requête** (requête dans une requête), le second select entre parenthèse, qui sert à calculer par regroupement (GROUP BY) le nombre de monuments par commune, et dont le résultat (T) est joint à la couche des communes, on peut ainsi utiliser le **champ nb** pour notre carte sur les communes.



Que se passent-ils pour les communes qui n'ont pas de monument ?

■ *Géométries dérivées*

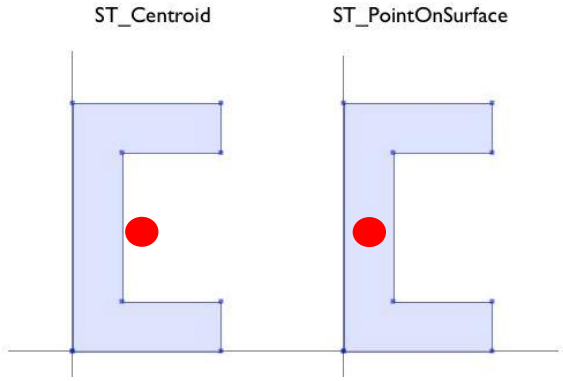
Cartographier des points sur les centroïdes des communes (polygones)



Polygone

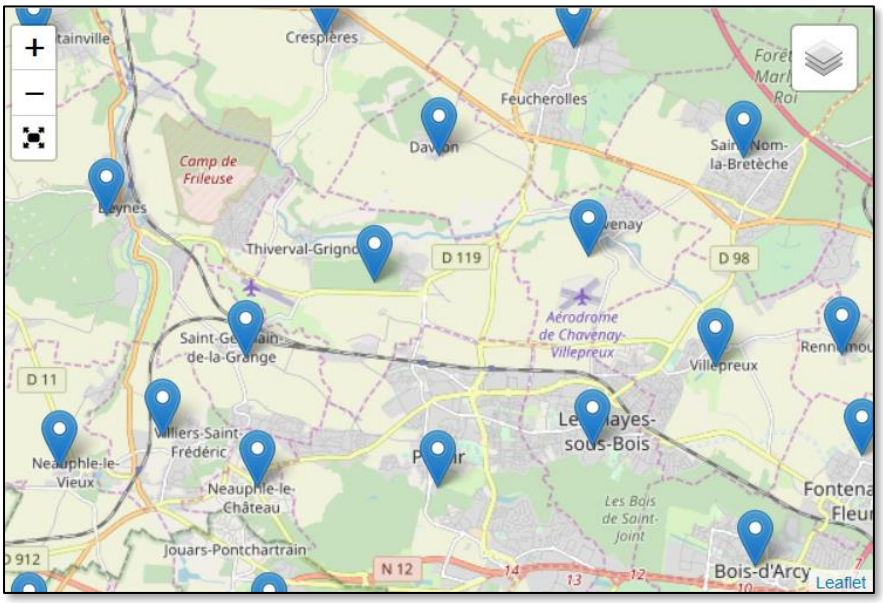
Point centroïde (centre de gravité)

ST_Centroid(geometry) ou **ST_PointOnSurface(geometry)**



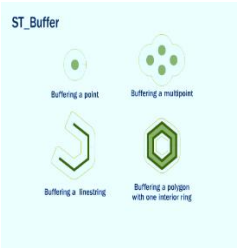

```
ST_AsGeoJSON(ST_Centroid(communes.geom))
```

Le résultat :



Cartographier des zones tampons (buffer) de 100 m autour des autoroutes

ST_Buffer(geom, 20)

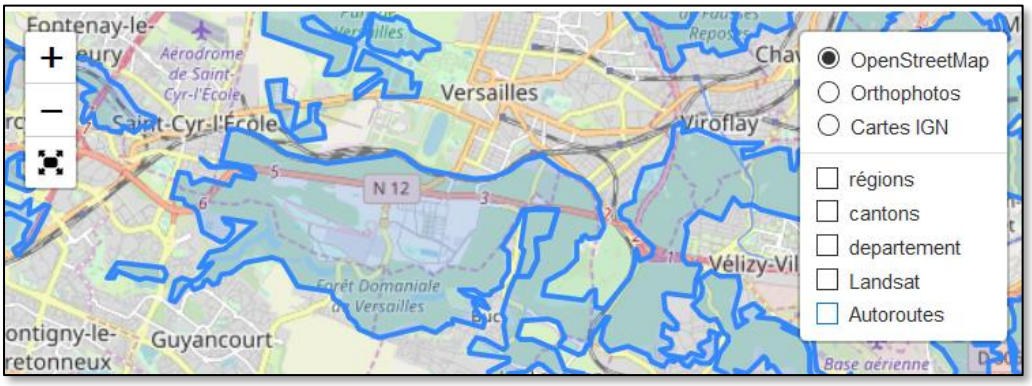


Le résultat :

■ *Intersection : croisement de couches*

Afficher les forêts et les autoroutes, mettez les autoroutes dans le contrôle de couches.

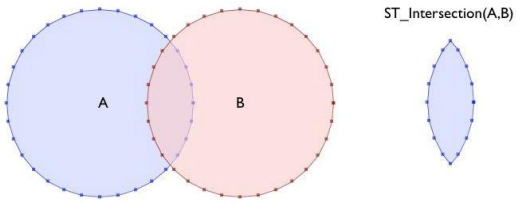
Regarder par exemple à Versailles, avec et sans afficher les autoroutes.



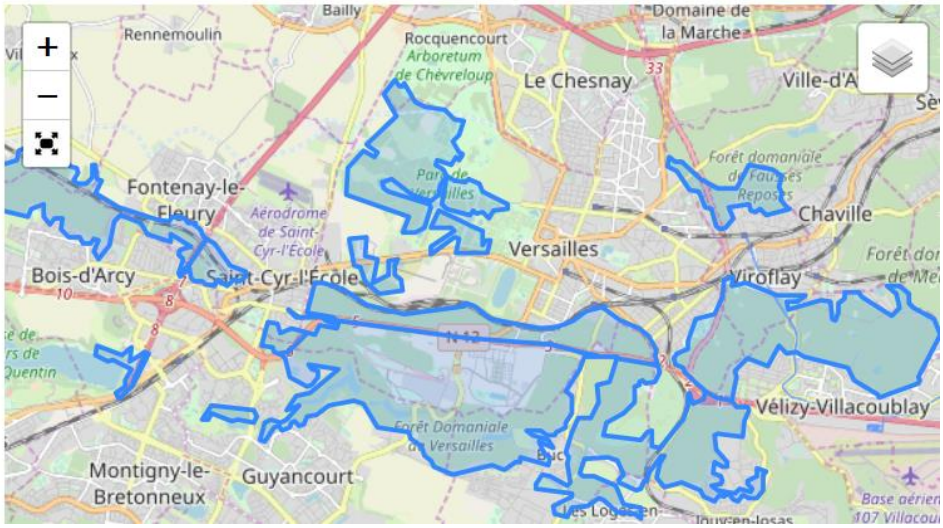
Le polygone de forêt n'est pas coupé par l'autoroute.

Faire le découpage des forêts par les autoroutes.

ST_Intersection(geometry A, geometry B) réalise un **croisement des couches A et B**



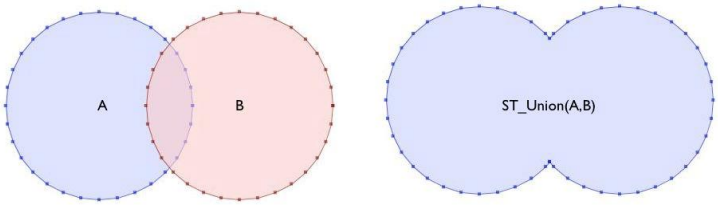
Résultat : le polygone de forêt qui était traversé par l'autoroute est maintenant coupé en 2.



■ *Fusion (Dissolve)*

Recréer les départements d'Ile de France par fusion des communes. (Fonction dissolve)

ST_Union(gA, gB) Fusionne gA et gB




```
SELECT departemen,ST_AsGeoJSON(ST_Union(geom)) as geojson from communes group by
departemen
```

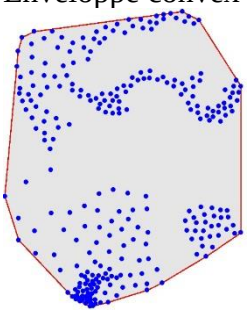
On réalise ici une fusion en fonction d'un critère commun aux communes (le champ departemen).

Le résultat :

■ *Enveloppe convexe ou concave*

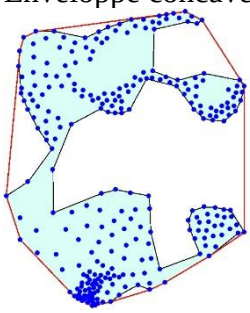
C'est le polygone « enveloppe » d'un jeu d'entités.

Enveloppe convex



ST_ConvexHull

Enveloppe concave

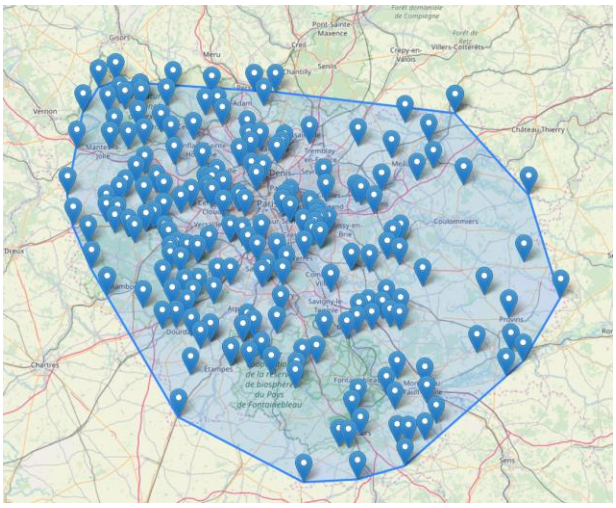


ST_ConcaveHull

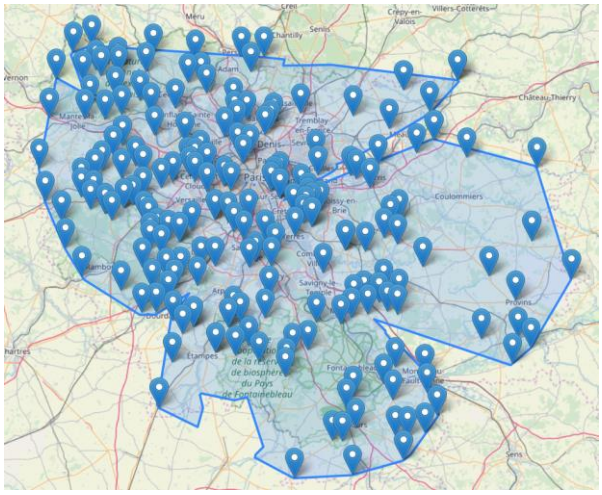
Construite des enveloppes convexe et concave des Châteaux.

```
SELECT ST_AsGeoJSON(ST_ConvexHull(ST_Collect(geom))) as geojson from
monuments where LEFT(monuments.tico,7)='Château'
```

```
SELECT ST_AsGeoJSON(ST_ConcaveHull(ST_Collect(geom),0.5)) as geojson
from monuments where LEFT(monuments.tico,7)='Château'
```



ST_ConvexHull



ST_ConcaveHull

IV. Exercice de synthèse : sondages pédologiques du plateau de Saclay

Le fichier SONDAGE_SACLAY.CSV est un extrait d'une base de données de sondages pédologiques du plateau de Saclay.

Les champs de la table :

NO_SONDAGE : identifiant du sondage
DATE : date de réalisation du sondage
LON : longitude (WGS84)
LAT : latitude (WGS84)
MORPHOLOGIE : description de la morphologie du terrain : plateau, vallée …
EXPOSITION : orientation de la pente.
BATTANCE : battance du sol
MAT_I : matériau du sol
OCSOL : occupation du sol
NOM_SOL : nom du sol
COMMUNE : commune du sondage

- Intégrer la table des sondages dans votre base POSTGIS en tant que table géographique.

- Créer une page Leaflet avec des couches de base en fond que l'on peut choisir dans un contrôle de couches

- Créer une couche overlay basique de vos points de sondage avec NO_SONDAGE comme étiquette

- Créer une couche overlay avec des symboles variables en fonction de l'occupation du sol.

- Demander à l'utilisateur le nom d'une commune, puis afficher une couche (zoomer dessus) qui ne contient que les sondages de cette commune.

- Afin de matérialiser les surfaces prospectées par la campagne pédologique, dans chaque commune, créer une zone tampon concave des sondages par commune.

ANNEXE 1 Autres modes d'importation de données géographiques dans POSTGIS

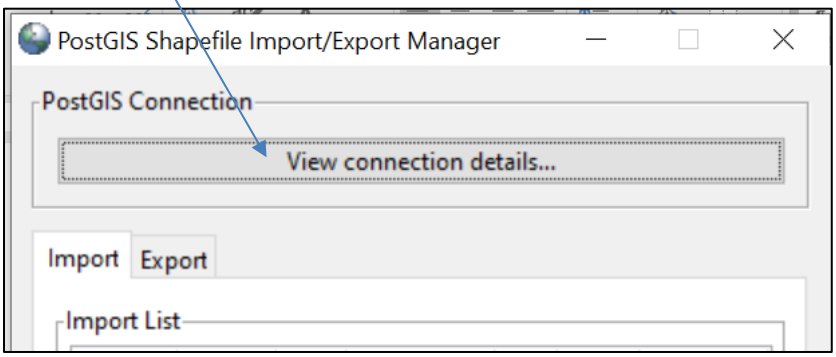
2.2 . Importation des shapefiles avec le logiciel Shp2pgsql-GUI

C'est une interface graphique à la commande précédente donc utilisable sans fenêtre de commande.

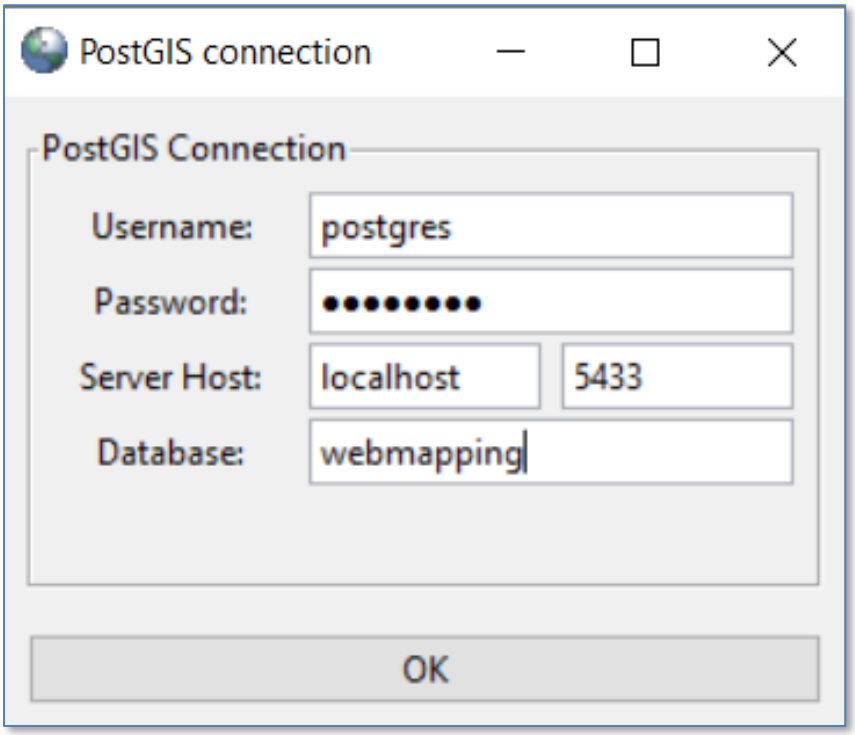
portablegis/postgresql/bin/postgisgui/ shp2pgsql-gui.exe

Importez les communes depuis le dossier BD_IDF dans la base de données WEBMAPPING de postgreSQL

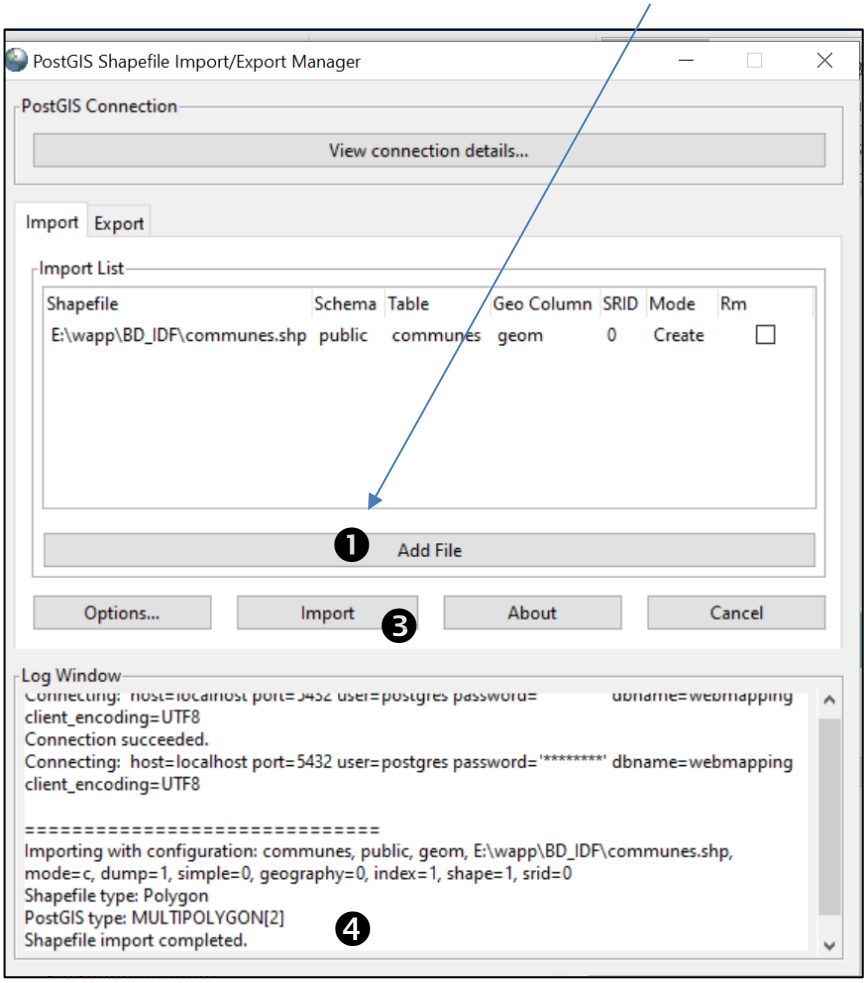
Connectez vous en cliquant sur « View connection details »



Entrez votre login et le nom de la base « webmapping »



Puis ajoutez votre couche des communes avec « Add File »



❷ Le fichier choisit apparait ici

❸ Bouton Importez

❹ Vérifiez le résultat est OK ici

2.3. Importation par la ligne de commande = autre méthode

Importation de couches SIG en format ShapeFile dans la base de données avec la commande en ligne shp2pgsql

⚠ Attention sous Mac OS X avec postgres.app, si cela n'a pas été fait au moment de l'installation, il faut inclure dans le chemin ($PATH) l'accès aux commandes en ligne (CLI) de postgres.

Avoir accès aux outils de commande en ligne (CLI Tools) de postgres :
Configurer le $PATH pour cela, dans un terminal de commande entrez la commande suivante puis valider par entrée:

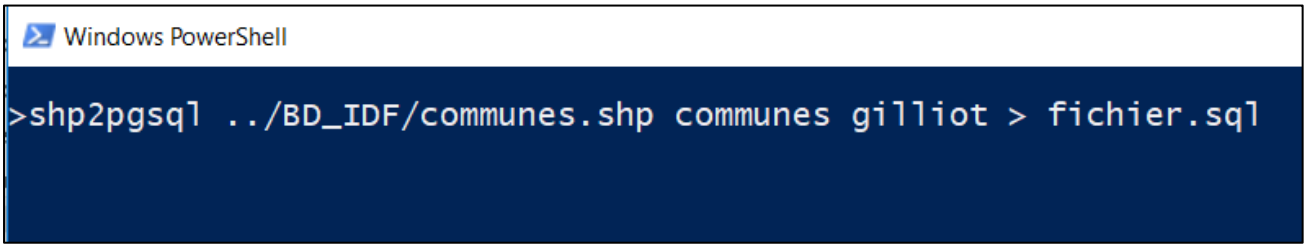
sudo mkdir -p /etc/paths.d &&
echo /Applications/Postgres.app/Contents/Versions/latest/bin | sudo tee /etc/paths.d/postgresapp

Fermer et relancer le terminal pour que les commandes soient accessibles.

La commande shp2pgsql pour convertir un fichier SHP en un fichier SQL que l'on peut alors importer dans la base de données postgreSQL

Sous windows la commande est dans le dossier bin de postgresql
communes.shp est la couche des communes d'Ile de France en format ShapeFile dans le dossier BD_IDF.

Exemple de commande :



communes est le nom de la table qui va être créée dans la base de données gilliot

Le symbole de redirection « > » envoi le résultat du calcul dans le fichier « fichier.sql »

Table des matières