Développement d'une application cartographique avec l'API Leaflet

Présentation de Leaflet[1]

Leaflet est une bibliothèque logicielle libre en JavaScript de cartographie interactive développée par Vladimir Agafonkin de Mapbox et de nombreux autres contributeurs.

La bibliothèque est utilisée sur les sites cartographiques OpenStreetMap (bibliothèque par défaut), Flickr, Wikipédia (greffon de cartographie et application mobile), Foursquare, Craigslist, Washington Post, le Wall Street Journal, Geocaching.com, City-Data.com, StreetEasy, Nestoria, Skobbler et beaucoup d'autres.

Elle peut être utilisée via une version en ligne dite *cdn* pour *content delivery network* ou téléchargée et utilisée en local. Pour un développement en local, il n'est donc pas nécessaire d'utiliser un serveur HTTP.

C'est une très bonne alternative à l'API Google Maps, qui bien que très fonctionnelle, présente un certain nombre de contraintes.

Historique

Depuis la version 0.6, sortie 26 juin 2013, cette bibliothèque utilise le format GeoJSON pour les structures de données géospatiales.

La version actuelle (publiée le 17/11/2019[2]) est la version 1.6.0.

Fonctionnalités

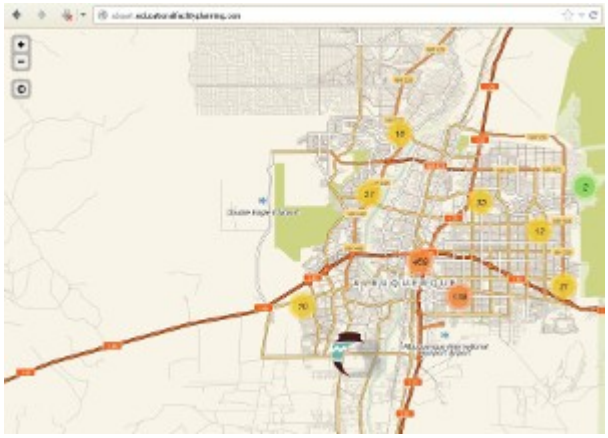
L'API est définie et ouverte et la bibliothèque supporte un système de greffons, permettant une grande extensibilité de l'API.

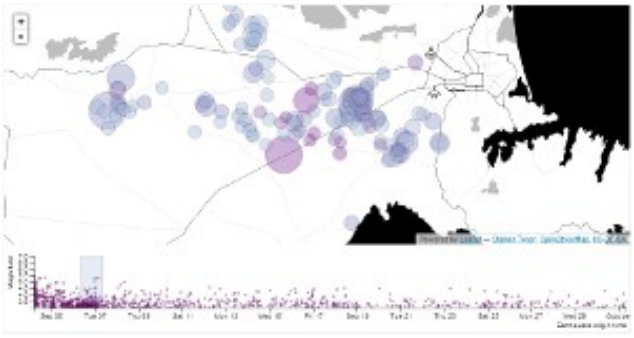
La bibliothèque supporte les calques WMS, GeoJSON , vectorielles et tuiles de façon native, et d'autres sont également supportées grâce au système de greffons.

Le code source est disponible sur GitHub : https://github.com/Leaflet/Leaflet

1 Présentation Wikipédia - http://fr.wikipedia.org/wiki/Leaflet
2 https://leafletjs.com/2019/11/17/leaflet-1.6.0.html

Exemples de réalisation





Programme

Durant ce TP, nous allons utiliser l'API cartographique JavaScript Leaflet pour créer une application Web cartographique sur les transports de la région Ile-de-France.

Données géographiques nécessaires

- Départements de la région Ile-de-France
- Positions des stations de Métro
- Réseau RER
- Réseau SNCF
- Stations Autolib

Logiciels nécessaires

- Editeur de texte avancé (Notepad++, SublimeText, EditPlus …)

Utilitaires

- Outils développeurs dans le navigateur (Firebug, Console JavaScript ...)
- QGIS
- Google Earth

Langages

- HTML
- CSS
- JavaScript

Liens utiles

- Le site officiel :
  http://leafletjs.com/
- Code-source du logiciel :
  https://github.com/Leaflet/Leaflet
- Vidéo du créateur de Leaflet - Vladimir Agafonkin - How Simplicity Will Save GIS
  http://vimeo.com/106112939
- Guide JavaScript :
  https://developer.mozilla.org/fr/docs/JavaScript/Guide
- Tester le JavaScript
  http://jsfiddle.net/

Exemples de réalisation

- http://www.openstreetmap.org
- http://chriswhong.com/open-data/taxi-techblog-2-leaflet-d3-and-other-frontend-fun/
- http://projectionwizard.org/

Sources pour les données

- http://wiki.openstreetmap.org/wiki/WikiProject_France/Transilien
- http://opendata.paris.fr/explore/dataset/stations_et_espaces_autolib_de_la_metropole_parisienne/?tab=export

Exercice pratique

Développement d'une application sur les transports en Île-de-France

Dans cet exercice, nous allons créer une application Web affichant une carte avec le réseau de transports en commun de la Région Ile-de-France, les stations de taxi, les bornes Autolib et d'autres données.

Avant toute chose, quelques recommandations utiles :

- Indenter et commenter le code.
- Utiliser les outils développeurs du navigateur, notamment la console JavaScript, pour tracer les erreurs.
- Distinguer l'affichage d'un fichier HTML local ou via le protocole http.
- Alléger et optimiser le code en créant des fonctions, des boucles, des constructeurs etc. ...
- Organiser les fichiers en créant des sous-répertoires pour les images (img), les styles (css) et les fichiers JavaScript (js).

Utilisation de l'API en ligne

Dans un nouveau dossier de travail *nom-prenom-tp-leaflet*, créer une nouvelle page html vierge, que vous nommez *index.html*.

Insérer le code suivant :

```
<!DOCTYPE html>
<html>
<head>
	<title>Transports en île-de-France</title>
	<meta charset="utf-8" />
	<meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>
</body>
</html>
```

Nous allons ajouter les appels à la librairie Leaflet qui est hébergée sur un serveur distant. Pour cela, insérer dans la partie head une balise script comprenant l'appel au fichier .js :

```
<script src="https://unpkg.com/leaflet@1.6.0/dist/leaflet.js"></script>
```

Nous allons aussi ajouter l'appel au fichier CSS distant :

```
<link rel="stylesheet" href="https://unpkg.com/leaflet@1.6.0/dist/leaflet.css" />
```

Et nous allons créer un nouveau div map au début de la partie body de la page html.

```
<div id="map"></div>
```

En CSS, dans une balise <style> définissez une hauteur en pixels pour la carte :

```
#map { height: 600px; }
```

Dans la partie body, créer une nouvelle balise de script pour appeler la carte avec un calque fourni par un service Web OpenStreetMap. C'est un des nombreux rendus cartographique des données OpenStreetMap. Leaflet va chercher un élément (map) du DOM. Ce code doit donc être placé après la balise déclarant le div map.

```
<script>
        var map = L.map('map').setView([48.75, 2], 8);
        L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png', {
        maxZoom: 18
        }).addTo(map);
</script>
```

Définition de l'emprise initiale de la carte

Changer le niveau de zoom et recentrer la carte sur la cité Descartes à Champs-Sur-Marne.

```
//définition de l'emprise initiale de la map
        map.setView([48.8392, 2.5876], 14);
```

Ajout de l'option Attribution sur l'objet L.tileLayer

```
attribution: 'Map data &copy; <a href="http://openstreetmap.org">OpenStreetMap</a>
contributors, '
```

L'attribution est importante, elle répond aux exigences des licences des données.

Ajout d'un calque GeoJSON

Nous allons ajouter nos propres données au format GeoJSON. Elles ont été exportées au format GeoJSON avec QGIS et sa fonction Exporter. Elles ont comme système de coordonnées le Spherical Mercator portant le code EPSG 3857.

Les données au format GeoJSON se présentent sous la forme suivante :

{ "type": "FeatureCollection",

"crs": { "type": "name", "properties": { "name": "urn:ogc:def:crs:OGC:1.3:CRS84" } },

"features": [{ "type": "Feature", "properties": { "color_qgis2leaf": '#438d25', "border_color_qgis2leaf": '#e23a00', "radius_qgis2leaf": 2.0, "transp_qgis2leaf": 0.33, "transp_fill_qgis2leaf": 1.0, "ID_GEOFLA": 76, "CODE_DEPT": "75", "NOM_DEPT": "PARIS", "CODE_CHF": "101", "NOM_CHF": "PARIS--1ER-ARRONDISSEMENT", "X_CHF_LIEU": 6516, "Y_CHF_LIEU": 68623, "X_CENTROID": 6517, "Y_CENTROID": 68620, "CODE_REG": "11", "NOM_REGION": "ILE-DE-FRANCE" }, "geometry": { "type": "MultiPolygon", "coordinates": [[[[2.416366869674164, 48.849247398441044], [2.415891919439672, 48.846636958382987], [2.416502442067038, 48.834705452668636], [2.422966688990253, 48.842714130214119], [2.427461322061275, 48.84163851119154], [2.437058573714734, 48.841171426217926],……..

…..Coordonnées géographiques des données GeoJSON……..

…….. , [2.243644299768495, 48.934337409495043], [2.230895517018339, 48.927862233855315], [2.227342046527635, 48.925438225947431], [2.220332556514582, 48.920527553643659], [2.216673090360718, 48.917958626317386], [2.200616497439371, 48.90879641122114]]]] } }

]

}

GeoJSON (de l'anglais Geographic JSON, signifiant littéralement JSON géographique) est un format ouvert d'encodage d'ensemble de données géospatiales simples utilisant la norme JSON (JavaScript Object Notation).

Il permet de décrire des données de type point, ligne, chaîne de caractères, polygone, ainsi que des ensembles et sous-ensembles de ces types de données et d'y ajouter des attributs d'information qui ne sont pas spatiale.

Le format GeoJSON, contrairement à la majorité des standards de systèmes d'informations géographiques, n'est pas écrit par l'Open Geospatial Consortium, mais par un groupe de travail de développeurs au travers d'internet.

QGIS permet de sauvegarder une couche de données au format GeoJSON. Pour les besoins du TP, nous disposons des couches de données au format GeoJSON suivantes :

Tableau des données

Couche	Nom du fichier	Description
Départements IDF	deps_idf.js	Départements de la région Ile-de-France
Rer A	rer_a.js	Ligne du RER A
Rer B	rer_b.js	Ligne du RER B
Rer C	rer_c.js	Ligne du RER C
Rer D	rer_d.js	Ligne du RER D
Rer E	rer_e.js	Ligne du RER E
Réseau SNCF national	reseau_sncf.js	Réseau ferré national
Gares SNCF IDF	gares_idf.js	Gares SNCF d'Ile-de-France
Stations AutoLib	stations_autolib.js	Stations Autolib

Pour les besoins du TP, les données sont préparées. Elles sont disponibles dans le dossier data.

Les chaînes geojson ont été placées dans une variable JavaScript et les fichiers ont été sauvegardés au format .js. Cette variable sera appelé sur la page principale pour traiter et afficher les données.

Commençons par appeler le fichier js de données :

```
<script src="data/rer_a.js" type="text/javascript"></script>
```

Ajoutons le code pour afficher la couche :

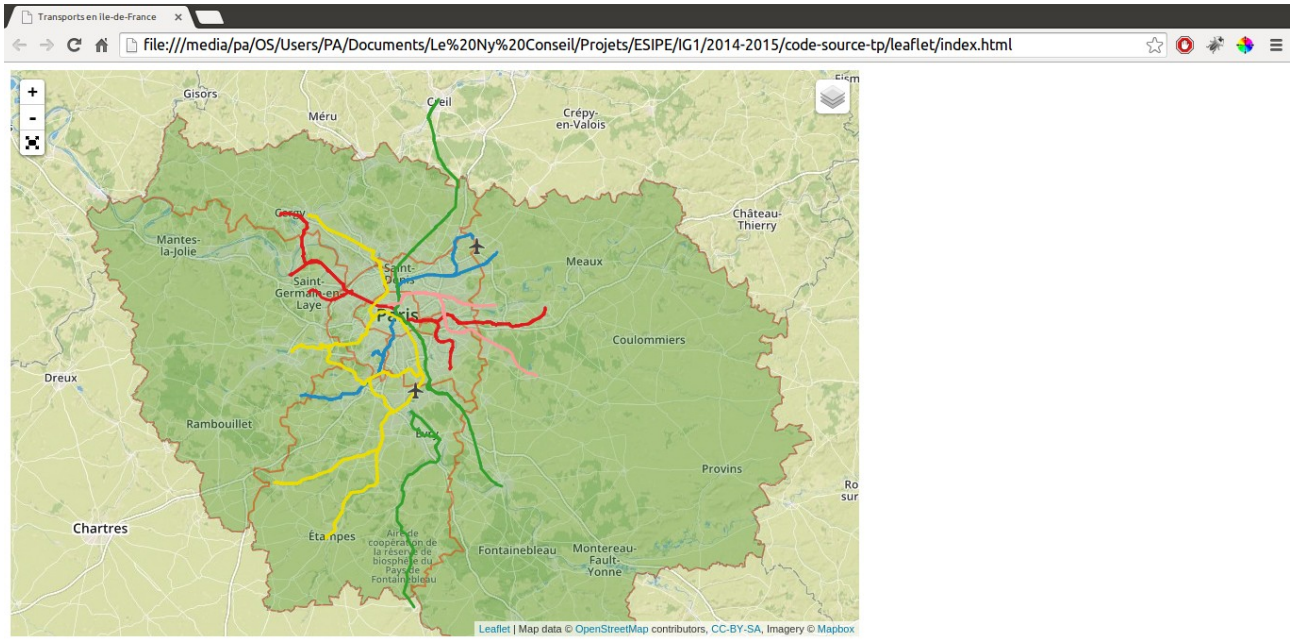
```
L.geoJson(RERA, {
//options possibles
}).addTo(map);
```

Pour notamment adapter le style, nous allons utiliser une autre notation pour ajouter un layer de données. Un exemple avec l'ajout de la couche RER D :

```
//Ajout du layer GeoJSON RER D
        var RERD = L.geoJson(RERD, { style: function (feature)
        {
        return {weight: feature.properties.radius_qgis2leaf,
```

```
color: feature.properties.color_qgis2leaf,
opacity: feature.properties.transp_qgis2leaf,
fillOpacity: feature.properties.transp_qgis2leaf};
}
                                        });
RERD.addTo(map);
```

Effectuer la même chose pour toutes les couches listées dans le tableau ci-dessus :



Ajout d'un marqueur

Nous allons ajouter des marqueurs pour afficher les aéroports avec un symbole personnalisé. L'image du symbole est stockée dans le dossier img et appelé depuis le fichier css.

```
var marker = L.marker([51.5, -0.09]).addTo(map);
```

Modifier la position du marqueur pour le placer sur Roissy-CDG.

Nous allons allons ajouter une image pour l'icône du marquer et nous allons grouper 3 points dans un groupe de couches :

```
//icone image pour POI Aéroport
```

```
var airportIcon = L.icon({
    iconUrl: 'img/icon_airport.png',
});
//Création d'un groupe de marqueurs pour les aéroports
var airports = new L.LayerGroup();

L.marker([48.7318763,2.369796], {icon: airportIcon}).bindPopup('Aéroport de Paris-Orly.').addTo(airports);
L.marker([49.009691,2.547924], {icon: airportIcon}).bindPopup('Aéroport de Paris-Roissy Charles-de-Gaulle.').addTo(airports);
L.marker([48.961472,2.437202], {icon: airportIcon}).bindPopup('Aéroport du Bourget').addTo(airports);

airports.addTo(map);
```

Changer la taille du marqueur :

```
iconSize: [40, 40], // size of the icon
```

Ajouter un mode plein écran

Nous allons ajouter une fonctionnalité permettant de basculer en mode plein-écran.
Nous commençons par ajouter un appels à un fichier JS dans la partie head de notre page.

```
<!-- appel script FullScreen -->
    <script src="js/Control.FullScreen.js"></script>
```

Nous allons également ajouter du code css directement dans la partie head :

```
<style type="text/css">
    #map { width: 95%; height: 600px; }
    .leaflet-control-zoom-fullscreen { background-image: url(img/icon-fullscreen.png); }
    /* on selector per rule as explained here : http://www.sitepoint.com/html5-full-screen-api/ */
    #map:-webkit-full-screen { width: 100% !important; height: 100% !important; z-index:
```

```
99999; }
                #map:-moz-full-screen { width: 100% !important; height: 100% !important; z-index: 99999; }
                #map:full-screen { width: 100% !important; height: 100% !important; z-index: 99999; }
                .leaflet-pseudo-fullscreen { position: fixed !important; width: 100% !important; height: 100% !important; top: 0px !important; left: 0px !important; z-index: 99999; }
                .fixed {
                position: fixed;
                top: 0;
                right: 50px;
                width: 250px;
                background-color: white;
                }

        </style>
```

Modifier l'instanciation de l'objet map pour lui passer les paramètres relatifs à la fonction :

```
//Instanciation de l'objet map
                var map = L.map('map', {
                        fullscreenControl: true,
                        fullscreenControlOptions: { // optional
                        title:"Show me the fullscreen !"
                        }
                });
```

Et, enfin, ajouter le code suivant :

```
        // detect fullscreen toggling
                map.on('enterFullscreen', function(){
                        if(window.console) window.console.log('enterFullscreen');
                });
                map.on('exitFullscreen', function(){
                        if(window.console) window.console.log('exitFullscreen');
                });
```

Ajouter une fonction de géocodage

Le géocodage permet de se localiser sur un point à partir d'une adresse.

Nous utiliserons un plugin Leaflet :

https://github.com/k4r573n/leaflet-control-osm-geocoder

Commencer par appeler le script suivant :

```
<!--appel script Geocoder -->
        <script src="js/Control.OSMGeocoder.js" type="text/javascript"></script>
```

Il faut également effectuer l'appel vers une feuille de style css spécifique :

```
        <!-- feuille de style geocoder-->
        <link rel="stylesheet"  type="text/css"  href="css/gecoder.css" />
```

enfin, afficher le contrôle dans le script principal de la partie *body* :

```
                //Ajout Geocoder
      var osmGeocoder = new L.Control.OSMGeocoder();
        map.addControl(osmGeocoder);
```

Interrogation des attributs

Nous allons utiliser la fonction native *bindpopup* pour afficher une fenêtre pop-up lors du clic sur un département :

Ajouter ce code dans le script principal

```
//fonction popup
                function onEachFeature(feature, layer) {
                        if (feature.properties) {
                         layer.bindPopup("<b>" + feature.properties.CODE_DEPT + "</b><br>" +
                                feature.properties.NOM_DEPT);
                                                }
```

```
                }
```

Et appeler la fonction onEachFeature danes les options de la couche deps_idf :

```
//Ajout du layer GeoJSON Departements IDF
                var deps_idf = new L.geoJson(deps_idf,{
                                        onEachFeature: onEachFeature,
style: function (feature) {
        return {color: feature.properties.border_color_qgis2leaf,
        fillColor: feature.properties.color_qgis2leaf,
        weight: feature.properties.radius_qgis2leaf,
        opacity: feature.properties.transp_qgis2leaf,
        fillOpacity: feature.properties.transp_qgis2leaf};
                        }
        });
```

Le résultat est visible lors d'un clic sur un département :

Afficher une barre d'échelle

Nous allons ajouter un contrôle affichant une barre d'échelle.

```
//Ajout de l'échelle
                L.control.scale().addTo(map);
```

Créer un groupe pour les RER.

Grouper les marqueurs

La couche des stations Autolib contient beaucoup trop d'informations pour être lisible.

Nous allons mettre en place le principe de clustering de markers pour grouper les marqueurs dans les zones denses.

Téléchargez le plugin MarkerCluster à l'adresse suivante :

https://github.com/Leaflet/Leaflet.markercluster/archive/v1.4.1.zip

Prenez le temps d'analyser son contenu. Nous allons commencer par appeler les fichiers js et css nécessaires. Copiez-les depuis le dossier *dist* de l'archive décompressée.

```
<!-- appel script marqueur cluster-->
        <script src="js/leaflet.markercluster-src.js"></script>
```

```
<!-- feuille de style marker cluster -->
        <link rel="stylesheet" href="css/MarkerCluster.css" />
        <link rel="stylesheet" href="css/MarkerCluster.Default.css" />
```

Ajout de titres et sous-titres directement sur la carte

Nous allons utiliser un contrôle Leaflet pour ajouter un calque de présentation de l'application.

```
var title = new L.Control();
                title.onAdd = function (map) {
                        this._div = L.DomUtil.create('div', 'info'); // create a div with a class "info"
                        this.update();
                        return this._div;
            };
            title.update = function () {
                        this._div.innerHTML = '<h2>This is the title</h2>This is the subtitle'
                };
                title.addTo(map);
```

Et le code CSS pour paramétrer le mise en forme des div avec la classe info

```
.info {
    padding: 6px 8px;
    font: 14px/16px Arial, Helvetica, sans-serif;
    background: white;
    background: rgba(255,255,255,0.8);
    box-shadow: 0 0 15px rgba(0,0,0,0.2);
    border-radius: 5px;
        }
    .info h2 {
    margin: 0 0 5px;
    color: #777;
}
```

Modifier le texte du calque par celui-ci :

```
this._div.innerHTML = '<h2>Les transports en <br> Ile-de-France </h2>2014<br><center><img
src="img/stif_logo.png" /><br><img src="img/sncf_logo.png" /><br><img
src="img/autolib_logo.png" /></center>'
```

Ajouter un autre calque OSM

Ajouter le layer Stamen Toner classique :

```
L.tileLayer('http://tile.stamen.com/toner/{z}/{x}/{y}.png', {
attribution: '&copy; <a href="http://maps.stamen.com/">Stamen Design</a>'
}).addTo(map);
```

Le définir en base layers. (le passer dans une variable) :

```
                //défintion des layers pour le contrôleur de couches

                var baseLayers = {
                        "OSM":osm,
                        "Toner":stamen,
                };
```

L'application doit maintenant proposer deux types de fond de plan et permettre de passer de l'un à l'autre.

Afficher maintenant un fond BING Maps Satellite.

http://www.microsoft.com/maps/create-a-bing-maps-key.aspx

Ajouter également un fond Google Maps Satellite

https://developers.google.com/maps/documentation/javascript/get-api-key

et d'autres fonds si souhaité :

http://tile.stamen.com/terrain/{z}/{x}/{y}.jpg

http://tile.stamen.com/watercolor/{z}/{x}/{y}.jpg

Ajout d'un fond WMS

Nous allons ajouter une couche provenant d'un serveur WMS :

http://leafletjs.com/reference.html#tilelayer-wms

```
var meteo = L.tileLayer.wms("http://mesonet.agron.iastate.edu/cgi-bin/wms/nexrad/n0r.cgi", {
layers: 'nexrad-n0r-900913',
format: 'image/png',
transparent: true,
attribution: "Weather data © 2012 IEM Nexrad"
});
```

Cette couche est une couche radar météo uniquement visible en Amérique du Nord.

Si vous aviez votre propre serveur OGC (Mapserver, GeoServer ou encore ArcGIS Server), vous pourriez ainsi exposer vos flux WxS et les lire ainsi dans votre application Leaflet.

Contrôleur de couches

Pour afficher un contrôleur de couches, qui va vous permettre de passer d'un *base layer* à un autre à l'aide de boutons radio, et pour afficher/masquer vos *overlays* avec des cases à cocher, vous devez vous référer à cette page présentant le *layer control* avec Leaflet :

https://leafletjs.com/examples/layers-control/

Geolocalisation

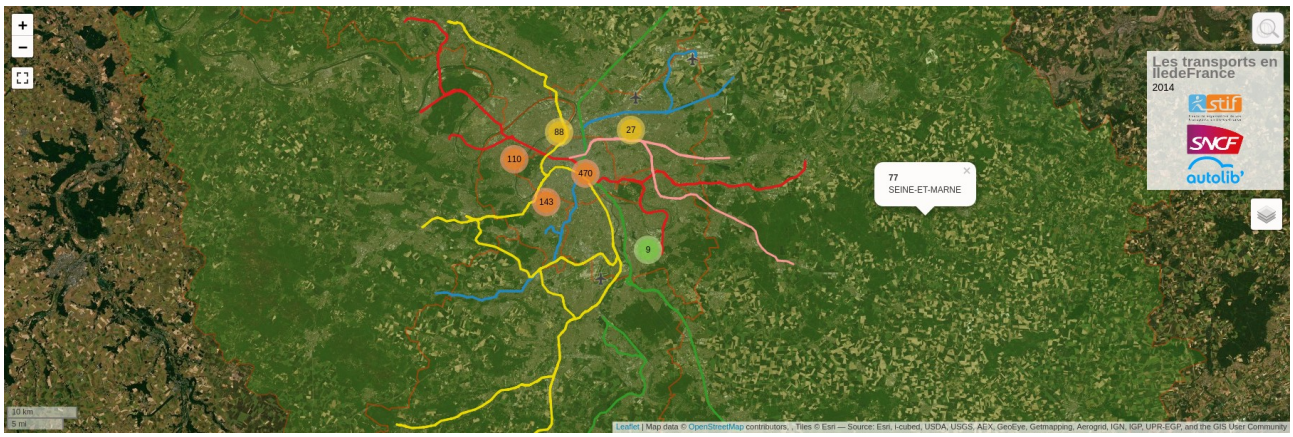
Utiliser et adapter les codes suivants pour offrir une fonctionnalité de géolocalisation :

```
map.locate({setView: true, maxZoom: 16});
```

```
function onLocationFound(e) {
var radius = e.accuracy / 2;
L.marker(e.latlng).addTo(map)
.bindPopup("You are within " + radius + " meters from this point").openPopup();
L.circle(e.latlng, radius).addTo(map);
}
map.on('locationfound', onLocationFound);
```

```
function onLocationError(e) {
alert(e.message);
}
map.on('locationerror', onLocationError);
```

Résultat final



Téléchargement de l'API et utisation d'une version locale

Télécharger et décompresser la version stable de la librairie Leaflet.
http://leafletjs.com/download.html

Adapter le code sur la page index pour pointer sur les ressources locales de Leaflet.

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

Effectuer la même opération pour les fichiers CSS et les autres fichiers JS.

Nettoyage des données

Nettoyer, commenter et optimiser votre code. Supprimer les ressources inutiles (fichiers data, img … etc.)

Transfert sur un serveur Web

Si vous disposez d'un espace web, copier le dossier de l'application dans votre dossier étudiant WWW ou un autre hébergement Web et afficher la page en utilisant le protocole HTTP.

Préparation du livrable

Déposer une archive zip nommée *nom_prenom_tp_websig.zip*. L'heure limite est l'heure de fin du TP.

Votre application doit être fonctionnelle en ouvrant la page index.html avec un navigateur moderne.

Utilisez une plateforme de dépôt pour partager votre livrable et envoyez un e-mail à l'adresse formation@lenyconseil.fr

Bonus

- Charger les données GeoJSON d'une autre manière de façon à charger directement les geojson (plutôt que dans une variable javascript) en utilisant le plugin Ajax Leaflet.

https://github.com/calvinmetcalf/leaflet-ajax

http://gis.stackexchange.com/questions/68489/how-to-load-external-geojson-file-into-leaflet-map