SHEVEK

Geste de savoir

Leaflet - Utilisation avancée

20 avril 2023

Table des matières

Introduction

Ce tutoriel est dédié à la présentation de fonctionnalités avancées de la bibliothèque Leaflet, il est dédié aux personnes connaissant déjà la bibliothèque. Pour ceux qui ne la connaissent pas et qui voudraient suivre ce tuto je vous conseille d'abord de suivre le tutoriel d'Eskimon .

Dans ce tuto nous allons apprendre à créer des interfaces utilisateur, à modifier les fonds de plans, à créer des marqueurs personnalisés et à ajouter des tooltips (texte) afin de créer des cartes interactives personnalisées.

1. Création d'une interface personnalisée

Introduction

Dans cette première partie nous allons voir comment l'on peut créer une interface personnalisée sur une carte Leaflet avec pour exemple une interface dotée d'un bouton avec une image à l'intérieur.

1.1. Afficher une carte

Pour démarrer on va créer une carte vide avec Leaflet, je vous mets directement à disposition le code, si vous n'avez jamais créé de carte avec Leaflet je vous conseille de suivre d'abord le tutoriel d'Eskimon

!(https://jsfiddle.net/y07fd94p/3/)

1.2. Créer une interface

Pour personnaliser nos cartes, Leaflet nous permet de créer des extensions de ses classes, c'est ce que nous allons faire pour créer notre interface, pour cela on va créer une extension la classe L.Control qui correspondra à notre interface.

Celle-ci contient:

— des options qui sont les propriétés de notre classe.
— une méthode onAdd exécutée à la création du composant.
— une méthode onRemove exécutée lors de sa destruction.

```
let MyControlClass =  L.Control.extend({

  options: {
  },

  onAdd: function(map) {
  },

  onRemove: function(map) {
  }
});
```

On crée ensuite une instance de notre nouvelle classe qu'on ajoute à la carte:

```
let myControl = new MyControlClass().addTo(map);
```

Notre interface est maintenant créée mais rien ne change à l'écran car elle est vide, pour afficher quelque chose il va falloir créer du contenu dans notre méthode `onAdd`, celle-ci est automatiquement appelée lorsque l'on exécute `addTo(map)`.

```
  onAdd: function(map) {
    var div = L.DomUtil.create('div', 'leaflet-bar my-control');

    div.innerHTML = 'Ma première interface';

    return div;
  },
```

Sur la première ligne on créer une nouvelle `div` html avec la méthode `L.DomUtil.create` qui prend pour paramètres:

- Le type d'élément HTML (div, img, button, ..)
- Les classes CSS dans notre exemple on a la classe `leaflet-bar` qui est une classe standard de leaflet et `my-control` qui est notre classe CSS personnalisée qui va nous permettre de définir notre design.

Ensuite on change le HTML de la div en y ajoutant du texte et pour que le contenu soit affiché on renvoie le contenu de notre div.

Pour améliorer le visuel, on ajoute un fond blanc à notre interface dans le css

```
.my-control
{
        background-color : white;
}
```

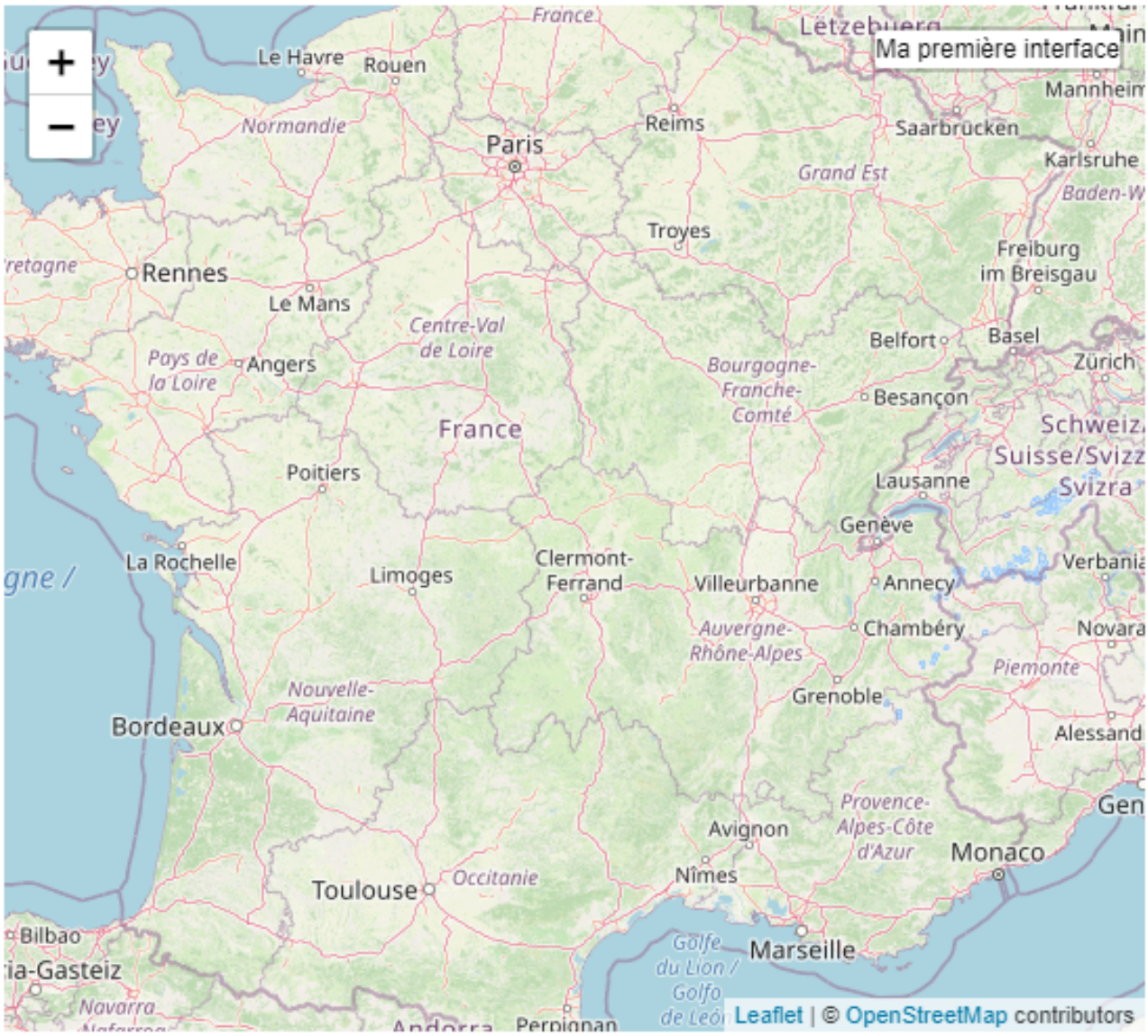


FIGURE 1.1. – On obtient notre première interface utilisateur

1.3. Positionnement

Les `options` des `L.Control` contiennent par défaut un paramètre `position` qui va nous permettre de choisir où sera placée notre interface sur l'écran, il y a 4 types de positions différentes:

- `topright`: En haut à droite
- `topleft`: En haut à gauche
- `bottomright`: En bas à droite
- `bottomleft`: En bas à gauche

Dans notre exemple on va positionner notre control en haut à gauche

```
options: {
    position: 'topleft'
},
```

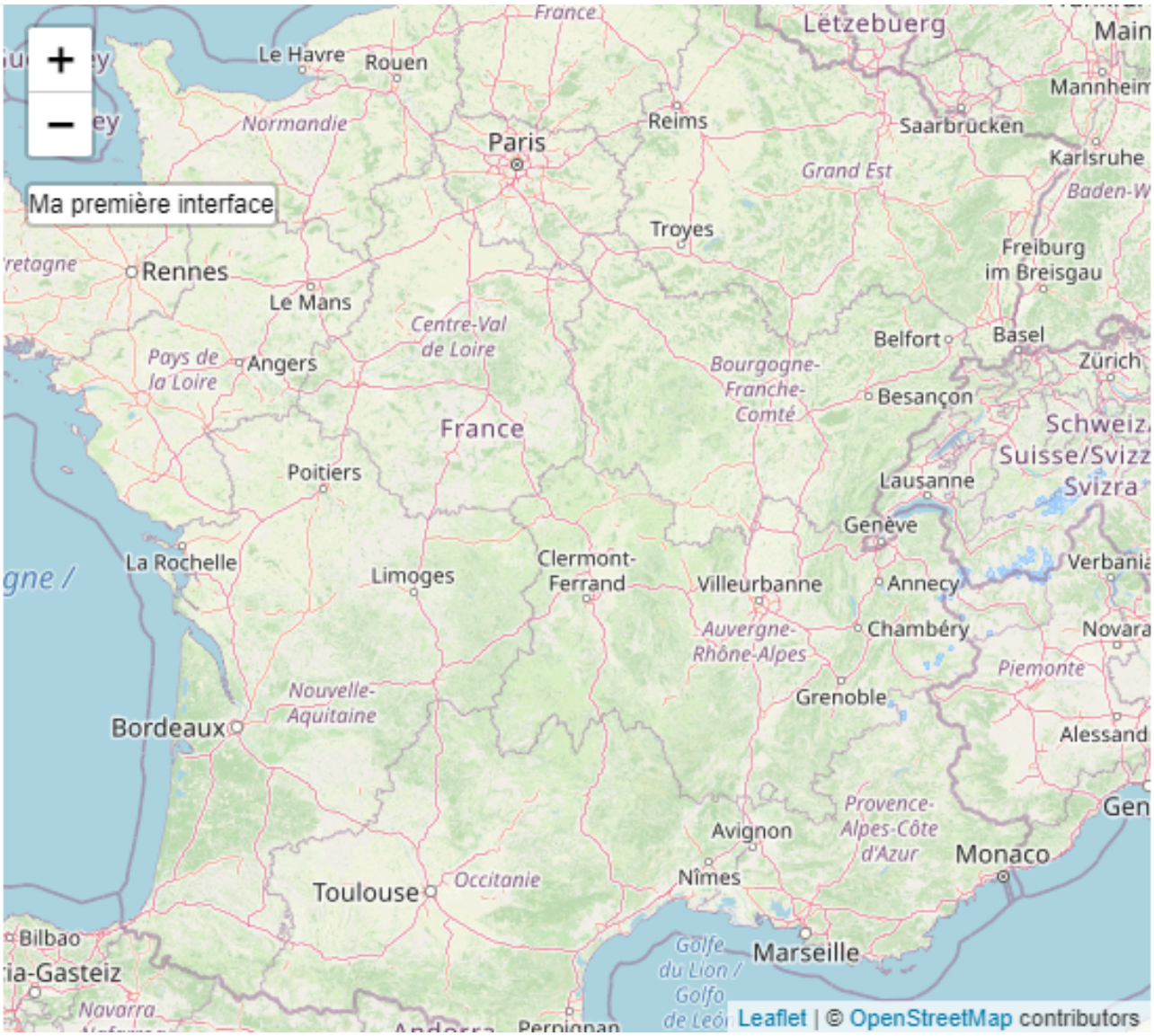


FIGURE 1.2. – Interface en haut à gauche

1.4. Afficher un bouton

On va maintenant créer notre premier bouton, sur notre interface, pour ajouter ce bouton on va utiliser à nouveau `L.DomUtil.create`, mais cette fois on va ajouter un troisième paramètre qui correspond à l'élément parent, dans notre cas il s'agira de l'élément `div` créé précédemment.

Code complet:

```
onAdd: function(map) {
  var div = L.DomUtil.create('div', 'leaflet-bar my-control');

  // Création de notre bouton et ajout à la div crée au-dessus
  var myButton = L.DomUtil.create('button', 'my-button-class',
      div);

  // Attribution d'un contenu HTML à notre bouton
  myButton.innerHTML = 'Mon button';

  return div;
},
```

1.5. Ajouter une icône

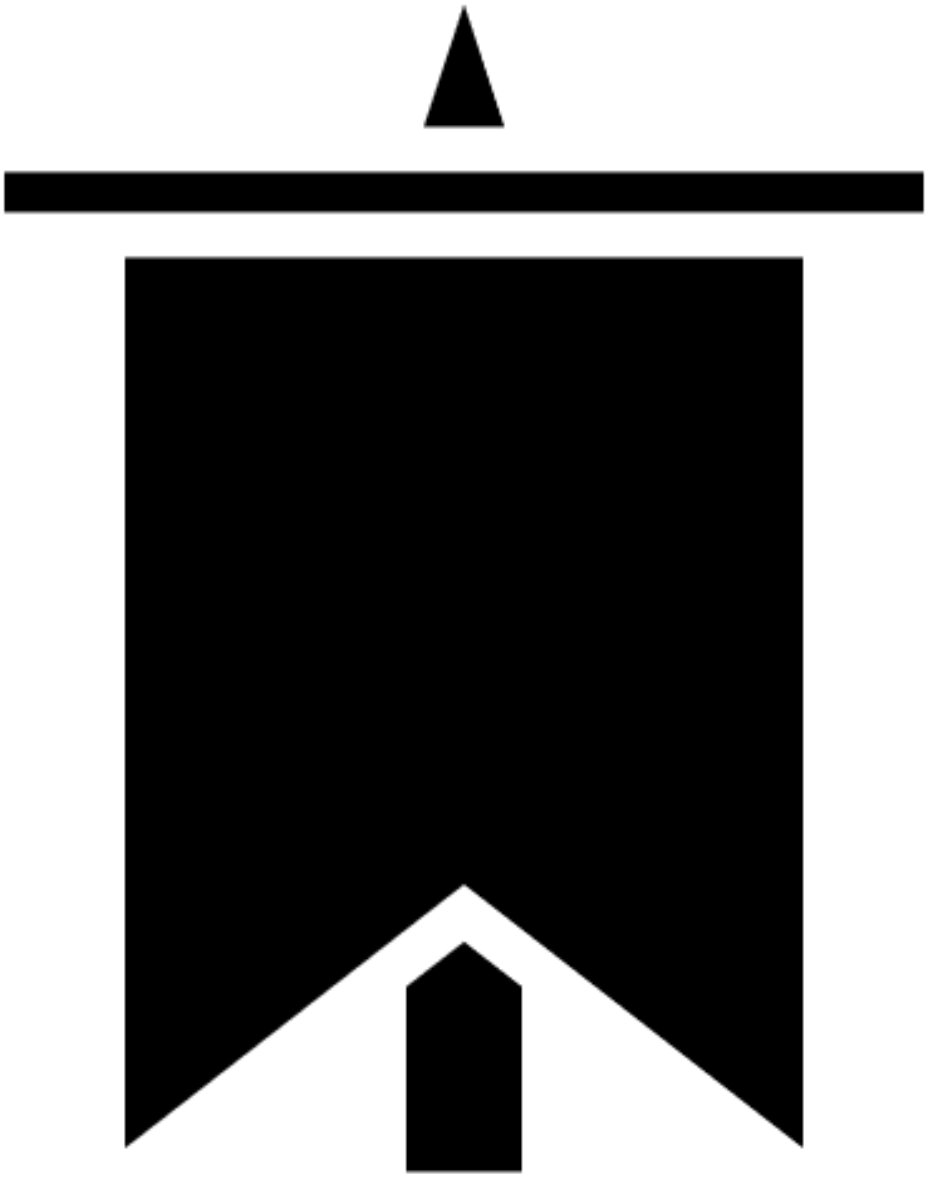


FIGURE 1.3. – banner.png

Nous allons ajouter l'image ci-dessus à notre bouton, pour cela la il faut la copier dans un sous-dossier appelé `img`, puis on ajoute notre image au bouton comme ceci:

```
1 onAdd: function(map) {
2     var div = L.DomUtil.create('div', 'leaflet-bar my-control');
3
```

```
4       var myButton = L.DomUtil.create('button', 'my-button-class',
            div);
5
6       let myImage = L.DomUtil.create('img', '', myButton);
7       myImage.src = "img/banner.png";
8       myImage.style = "margin-left:0px;width:20px;height:20px";
9
10      return div;
11    },
```

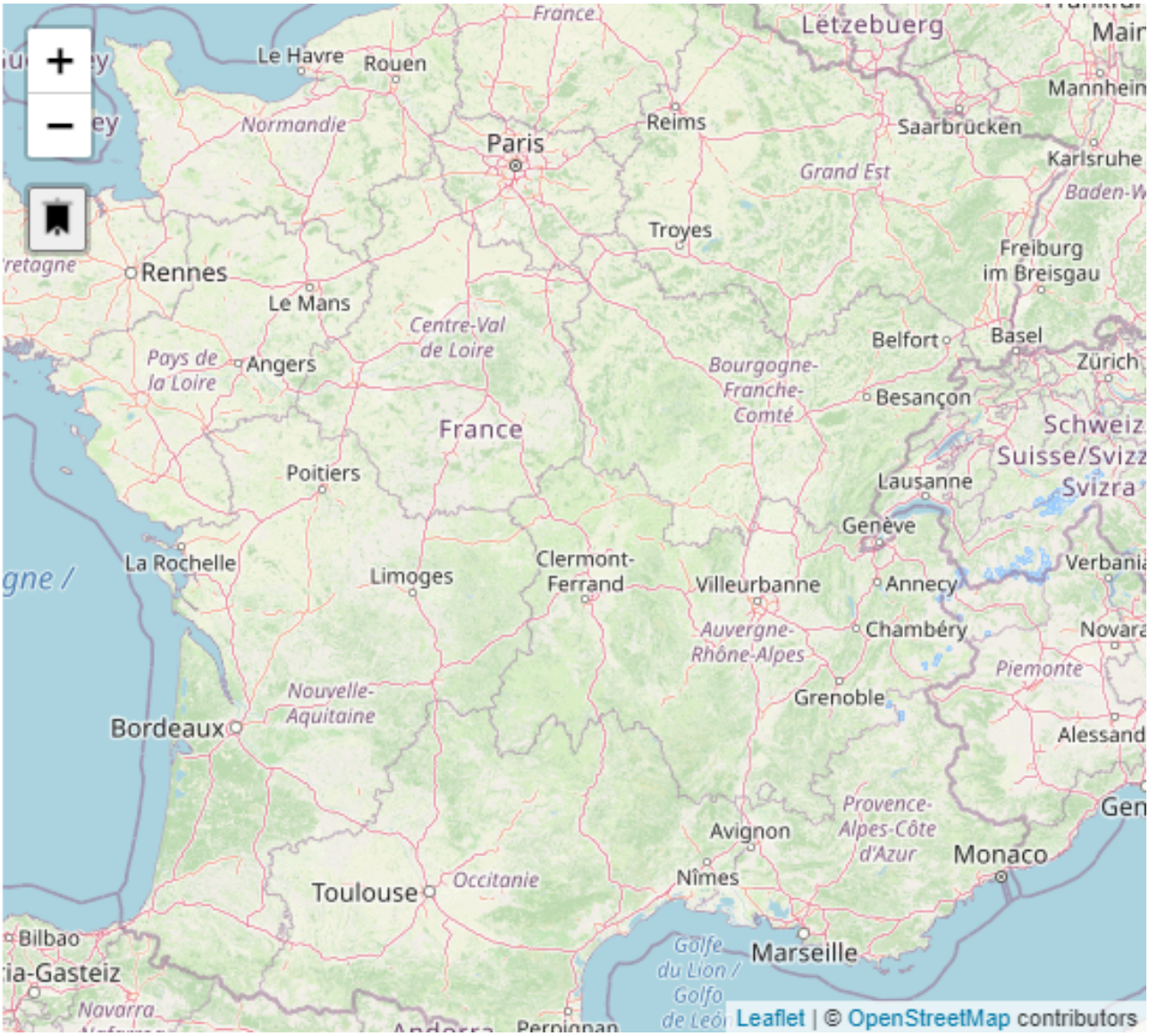


FIGURE 1.4. – image.png

1.6. Gerer les actions sur notre bouton

Il nous reste à gérer les évènements sur notre bouton, on va ajouter une alerte au clic, pour cela on utilise `L.DomEvent.on` qui permet d'écouter les évènements du DOM avec pour paramètres:

— L'élément cible, dans notre cas le bouton que l'on vient de créer.
— le type d'action, dans notre cas une action de clic.
— La fonction à appeler lors du clic, pour nous une fonction simple qui affiche une alerte.
— Le contexte, on utilise le mot-clé `this` par défaut.

```
1 L.DomEvent.on(myButton, 'click', function() {
      alert("clic sur le button"); }, this);
```

Récapitulatif du code:

!(https://jsfiddle.net/ktvuxwd9/3/)

Conclusion

Maintenant que nous avons notre première interface, on va y ajouter des actions concrètes dans une seconde partie avec le changement de fond de plan.

2. Gérer les fonds de plans

Introduction

Dans cette partie nous allons utiliser notre interface utilisateur pour pouvoir changer de fond de plan.

2.1. Qu'est ce qu'un fond de plan

Les tuiles de fond de plan correspondent à des images qui sont affichées pour représenter la carte du monde, il en existe plusieurs types notamment des cartes où on voit les villes et les routes ou des cartes satellites. Par défaut nous avons utilisé OpenStreetMap mais sachez qu'il en existe beaucoup d'autres qui sont répertoriés ici .

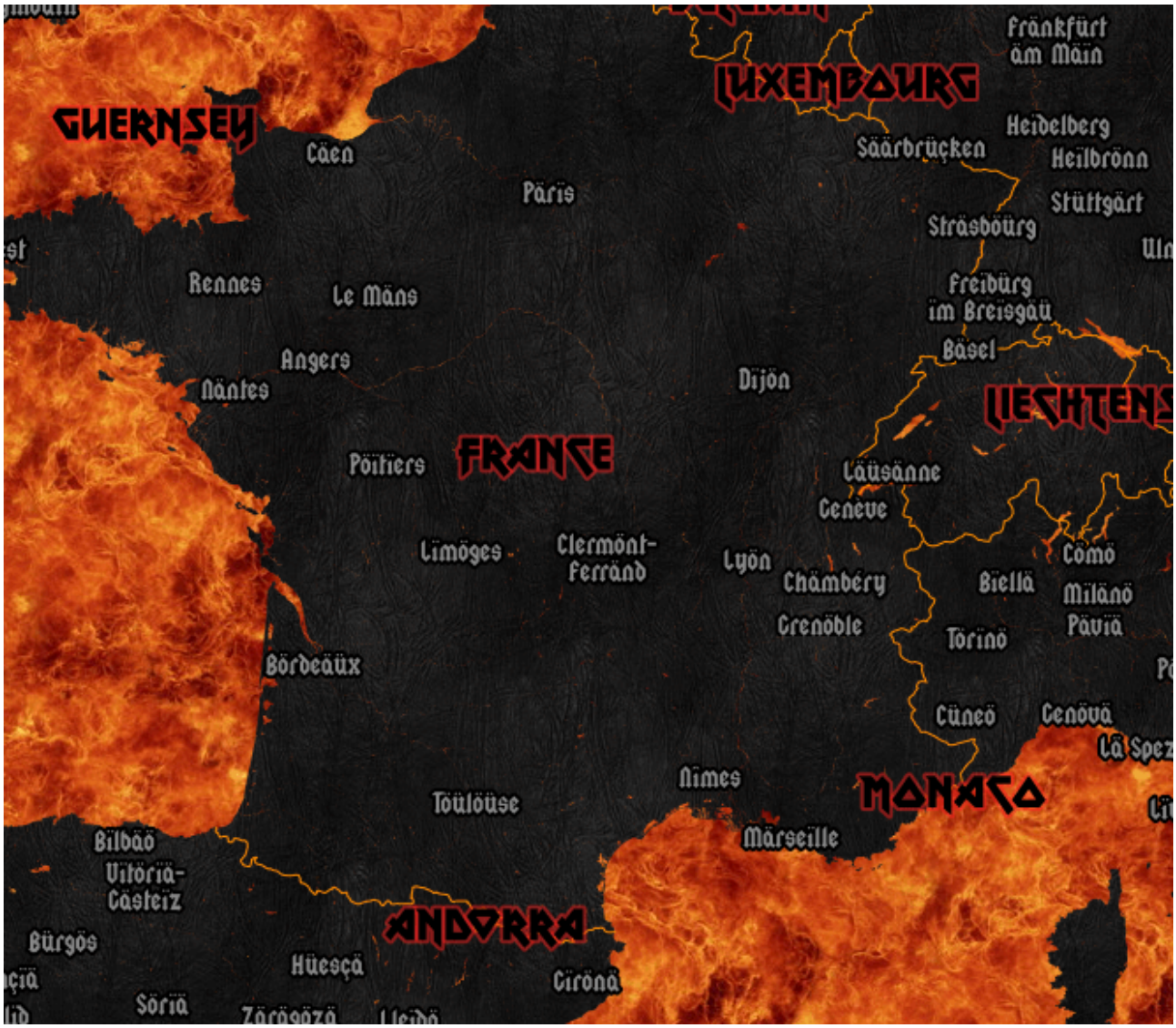


FIGURE 2.1. – Un fond de plan très Métal :)

2.2. Utiliser un fond de plan satelite

Dans notre précédent chapitre nous utilisons le fond de plan d'OpenStreetMap comme ceci:

```
1 L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png', {
2     attribution:
          '&copy; <a href="http://osm.org/copyright">OpenStreetMap</a> contributo
3 }).addTo(map);
```

Si nous utilisons à la place ceci:

```
L.tileLayer(
    'http://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/MapServe
    ,
    {
    attribution: 'ArcGIS'
}).addTo(map);
```

Nous obtenons une carte avec un fond de plan satellite fournie par ArcGis

2.3. Action de changement de fond de plan

Nous allons maintenant ajouter un bouton pour changer dynamiquement de fond de plan, la première étape va être d'ajouter un nouveau bouton avec l'icône ci-dessous.

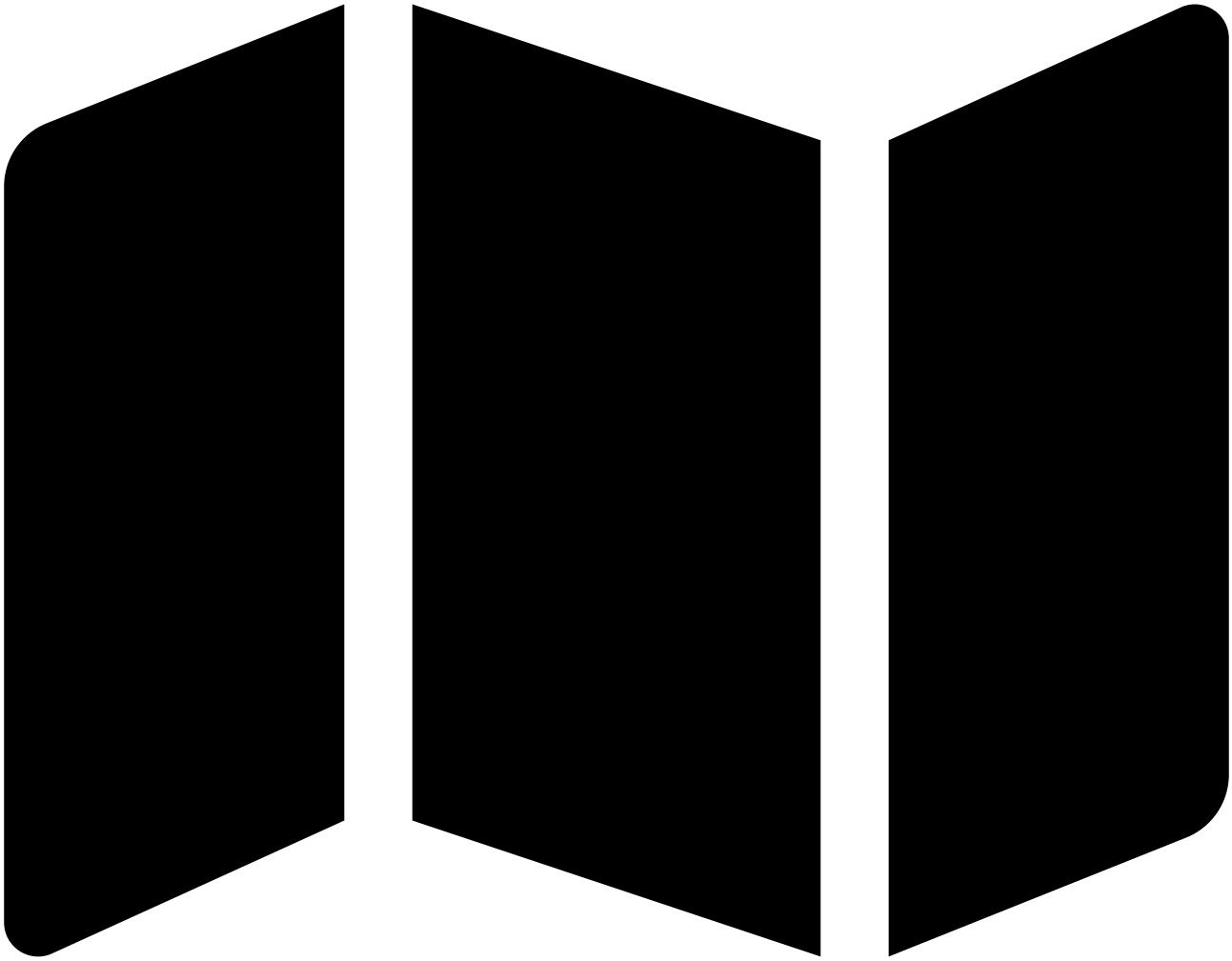


FIGURE 2.2. – map-solid.svg

D'abort on change d'initialisation de notre tile et on définie une variable qui nous permettra de savoir sur quel tile on est actuellement.

```
let selectedTile =
    L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png', {
    attribution:
        '&copy; <a href="http://osm.org/copyright">OpenStreetMap</a> contributo
}).addTo(map);

let tileType = "OpenStreetMap";
```

On crée un nouveau bouton comme dans le chapitre précédent qui appelle une méthode `changeBackground` :

```
    var buttonBackground = L.DomUtil.create('button',
        'my-button-class', div);

    let myBackground = L.DomUtil.create('img', '',
        buttonBackground);
    myBackground.src = "img/map-solid.svg";
    myBackground.style = "margin-left:0px;width:20px;height:20px";

   L.DomEvent.on(buttonBackground, 'click', function() {
       this.changeBackground(); }, this);
```

On crée notre méthode `changeBackground`qui met à jour le layer selon le type précédent de layer

```
  changeBackground()
  {
    this.map.removeLayer(selectedTile);

    if(tileType == "OpenStreetMap")
    {
      tileType = "ArcGis";

      selectedTile = L.tileLayer(
          'http://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/Ma
          ,
          {
        attribution: 'ArcGIS'
      }).addTo(this.map);
    }
    else
    {
      tileType = "OpenStreetMap";

      selectedTile =
          L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png', {
        attribution:
            '&copy; <a href="http://osm.org/copyright">OpenStreetMap</a> contr
      }).addTo(map);
    }
  },
```

Enfin on modifie le CSS de nos boutons pour avoir un bon affichage

```
.my-button-class
{
        padding : 2px;
        display : block;
        border-radius : 0px;
}
```

!(https://jsfiddle.net/5fupw61k/4/)

Conclusion

Maintenant que nous savons comment changer de fond de plan, nous allons créer et personnaliser des marqueurs.

3. Ajouter de marqueurs

Introduction

Leaflet met à disposition plusieurs manières de créer des marqueurs :

- Des marqueurs simples.
- Des marqueurs avec une icône.
- Des marqueurs à partir de code HTML.
- Mais aussi de nombreux plugins permettant de créer d'autres types de marqueurs.

3.1. Afficher un marqueur basique

Pour ajouter un marqueur on crée un objet L.marker avec une position qu'on ajoute à la carte.

```
1 let marqueur = L.marker([48.856944, 2.351389]).addTo(map)
```

Plus d'informations sur le tuto d'Eskimon

Sur notre marqueur on peut ajouter un pop-up qui s'ouvrira au clic de la souris.

```
1 marqueur.bindPopup('Ceci est un test');
```

Ou directement à la création du marqueur:

```
1 let marqueur = L.marker([48.856944,
      2.351389]).bindPopup('Ceci est un test').addTo(map)
```

3.2. Afficher un marqueur avec une icône personnalisées

Pour créer des marqueurs avec des images personnalisées, nous allons utiliser des `L.icon` permettant de paramétrer des icônes personnalisés.

Il nous faut créer une extension de la classe L.Icon sur laquelle on définit des options de taille (iconSize), de décalage de l'icône (iconAnchor) et de décalage de la popup (popupAnchor)

```
var myIconClass = L.Icon.extend({
    options: {
        iconSize:     [38, 95],
        iconAnchor:   [22, 94],
        popupAnchor:  [-3, -76]
    }
});
```

On crée une instance de notre classe, dans laquelle on définit ensuite l'image de l'icône

```
var icon = new myIconClass ({
    iconUrl:
        'https://zestedesavoir.com/media/galleries/16186/d2a9ef14-71b2-4acb-a1
});
```

Enfin on crée le marqueur avec notre icône

```
L.marker([48.856944, 2.351389], {icon:
    icon}).bindPopup('Ceci est un test').addTo(map);
```

On peut également définir des ombres à notre image, pour cela on définit la taille, la décalage de l'ombre et on ajoute notre image d'ombre.

```
var myIconClass = L.Icon.extend({
    options: {
        iconSize:     [20, 20],
        iconAnchor:   [10, 10],
        popupAnchor:  [0, -10],
        shadowSize:   [20, 20],
        shadowAnchor: [6, 8],
    }
});

var icon = new myIconClass({
```

```
12      iconUrl: 
            'https://zestedesavoir.com/media/galleries/16186/d2a9ef14-71b2-4acb-a1
            ,
13      shadowUrl: 
            'https://zestedesavoir.com/media/galleries/16186/3503ba88-d052-4f5f-81
14  });
```

FIGURE 3.1. – Notre marqueur personnalisé avec une ombre

3.3. Ajout d'un marqueur au clic

Nous allons maintenant faire un système d'ajout d'un nouveau marqueur au clic de la souris, pour cela nous allons ajouter un bouton et un système d'ajout de marqueur.

Dans la méthode `onAdd`, on ajoute notre bouton:

```
1     var buttonAddMarker = L.DomUtil.create('button',
          'my-button-class', div);
2
3      let backgroundAddMarker = L.DomUtil.create('img', '',
          buttonAddMarker);
4      backgroundAddMarker.src = 
          "https://zestedesavoir.com/media/galleries/16186/d2a9ef14-71b2-4acb-a1
          ;
5      backgroundAddMarker.style =
          "margin-left:0px;width:20px;height:20px";
6
7      L.DomEvent.on(backgroundAddMarker, 'click', function() {
          this.addMarker(); }, this);
```

Pour gérer le clic sur la carte il faut écouter l'évènement `mousedown` de notre map, il prend en paramètre un évènement sur lequel on peut récupérer la latitude et la longitude du clic.

```
1  let enableAddMarker = false;
2  map.on('mousedown', function(e)
3  {
4    if(enableAddMarker)
5    {
```

```
    enableAddMarker = false;
    L.marker([e.latlng.lat, e.latlng.lng], {icon:
        icon}).bindPopup('Ceci est un test').addTo(map);
  }
});
```

Enfin au clic sur le bouton activation, on met à jour la variable d'ajout de marqueur

```
  addMarker()
  {
    enableAddMarker = true;
  }
```

Code complet:

!(https://jsfiddle.net/5fupw61k/5/)

3.4. Marqueur HTML et plugins

Il est possible de créer des marqueurs plus personnalisés, si vous le souhaitez je vous laisse suivre ce tuto (en anglais)

Il existe également beaucoup de modules permettant de personnaliser ses marqueurs :

- Un module que j'ai réalisé afin de pouvoir afficher des icônes colorisées (à partir d'images noires)
- Un module pour créer de belles icônes à partir de Glyphicons ou Font-Awesome

Conclusion

Dans la prochaine partie nous verrons comment créer des tooltips personnalisés.

4. Ajouter de tooltips

Introduction

Les tooltips nous permettent d'ajouter des zones de texte sur notre carte afin d'afficher un nom de lieu, un évènement ou d'autres infos.

4.1. Créer un tooltip

Pour créer un tooltip on utilise `L.tooltip` avec comme paramètres les options de création (un objet vide dans notre exemple).

```
let tooltip = L.tooltip({});
```

On doit définir également sa position:

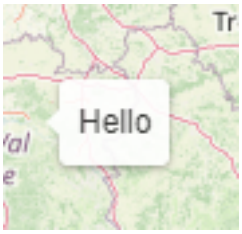
```
tooltip.setLatLng(new L.LatLng(47.856944, 2.351389));
```

On ajoute le contenu à afficher dans le tooltip:

```
tooltip.setContent(`Hello`);
```

Enfin on l'ajoute à la carte:

```
tooltip.addTo(map);
```



4.2. Modifier les options de notre toolitp

La première façon de personnaliser notre tooltip est de modifier ces paramètres. (Documentation)

Option	Type	Valeur par défaut	Description
pane	String	'tooltipPane'	Couche de la carte où l'info-bulle sera ajoutée, par defaut il est afficher sur un couche dédié au tooptip.
offset	Point	Point(0, 0)	Décalage facultatif de la position de l'infobulle.
direction	String	'auto'	Direction de l'infobulle. Les valeurs possibles sont : droite, gauche, haut, bas, centre, auto. auto basculera dynamiquement entre la droite et la gauche selon la position de l'infobulle sur la carte.
permanent	Boolean	false	Définie si l'infobulle doit être ouverte en permanence ou seulement au passage de la souris.
sticky	Boolean	false	Si true, l'infobulle suivra la souris au lieu d'être fixée au centre de l'élément.
interactive	Boolean	false	Si true, l'infobulle écoutera les événements.
opacity	Number	0.9	Opacité de l'info-bulle.

FIGURE 4.1. – Options des L.tooltip

Par exemple, on choisit de mettre le label au centre, on le rend permanent (ne disparaît pas au clic) et on lui attribue une transparence (opacity).

```
let tooltip = L.tooltip({
  direction: 'center',
  permanent: true,
  opacity: 0.6
});
```

4.3. Editer le html et css de notre tooltip

Pour avoir un tooltip personnalisé, on peut également modifier son code CSS et son HTML.

Pour modifier le css on peut ajouter une classe CSS via l'attribut className du tooltip (on remet également l'opacité à 1):

```
let tooltip = L.tooltip({
  direction: 'center',
```

```
  permanent: true,
  opacity: 1,
  className: 'tooltip'
});
```

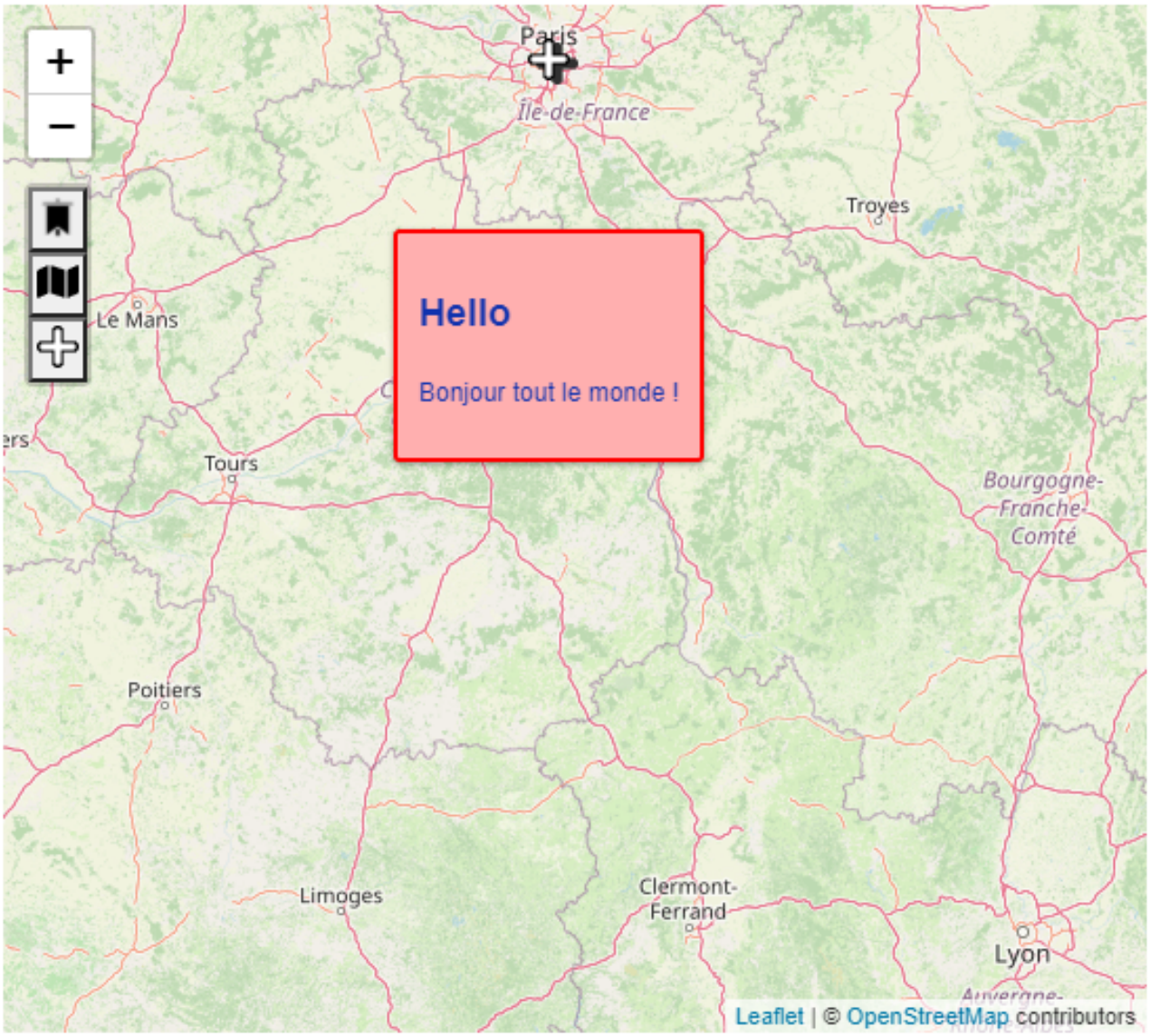
Ensuite on peut déterminer le CSS de la classe définie, avec pour exemple la couleur du texte en bleu, la couleur de fond en rouge, un padding de 10px et une bordure rouge.

```
.tooltip
{
  background-color: #ffafaf;
  color : #1032ae;
  padding: 10px;
  border : 2px solid red;
}
```

On obtient un carré rouge avec du texte en bleu:

Dans le contenu du tooltip, on peut également écrire du code HTML afin de personnaliser notre tooltip.

```
tooltip.setContent(`<h2>Hello</h2><p>Bonjour tout le monde !</p>`);
```



Conclusion

Cette partie sur les tooltip est terminée, au cas où vous voudriez afficher des tooltips sur des positions proches mais que ne se chevauchent pas ce plugin pourrait vous être utile.

5. Filtrer des données

Introduction

Dans cette dernière partie on va apprendre à filtrer des données, pour cela on va faire une petite interface nous permettant de filtrer les données provenant d'un fichier GEOJSON.

Pour nous allons partir de données GEOJSON avec des points correspondant à des évènements avec pour paramètres:

— Une date (année)
— Un type (”AUTRE”, ”PUBLIC” ou ”COMMERCE”)

Avec notre interface nous allons filtrer:

— La date minimale des évènements
— La date maximale des évènements
— Les types (3 choix: ”AUTRE”, ”PUBLIC” ou ”COMMERCE”)

5.1. Charger des données GeoJSON

Pour nos données on va utiliser le format GEOJSON qui est le format le plus largement utilisé pour les systèmes de cartographies web.

`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).` - source Wikipédia

On va d'abort initialiser la carte, créer un objet GEOJSON et charger ces données et les ajouter à la carte:

```
1 // Initialisation de la carte
2 let map = L.map('macarte').setView([40.722998, -74.003949], 13);
3
4 let selectedTile =
      L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png', {
5     attribution:
          '&copy; <a href="http://osm.org/copyright">OpenStreetMap</a> contributo
6 }).addTo(map);
7
8 // Creation de l'objet geojson
```

```
let geojsonFeature = {
    "type": "FeatureCollection",
    "features": [{
        "type": "Feature",
        "properties": {
            "shape": "Marker",
            "name": "Unnamed Layer",
            "category": "default",
            "year": 2022,
            "type": "COMMERCE"
        },
        "geometry": {
            "type": "Point",
            "coordinates": [-74.003949, 40.722998]
        },
        "id": "7c587f8f-6ad8-4638-b831-cb368b3a598d"
    }, {
        "type": "Feature",
        "properties": {
            "shape": "Marker",
            "name": "Unnamed Layer",
            "category": "default",
            "year": 2022,
            "type": "PUBLIC"
        },
        "geometry": {
            "type": "Point",
            "coordinates": [-74.002147, 40.719876]
        },
        "id": "2367c632-4ce1-4b4e-b705-1f7698dc70d0"
    }, {
        "type": "Feature",
        "properties": {
            "shape": "Marker",
            "name": "Unnamed Layer",
            "category": "default",
            "year": 2012,
            "type": "AUTRE"
        },
        "geometry": {
            "type": "Point",
            "coordinates": [-73.998027, 40.722413]
        },
        "id": "0aade070-1667-4cb2-b22b-b6db4215453d"
    }, {
        "type": "Feature",
        "properties": {
            "shape": "Marker",
            "name": "Unnamed Layer",
            "category": "default",
```

```
            "year": 2015,
            "type": "AUTRE"
        },
        "geometry": {
            "type": "Point",
            "coordinates": [-74.008799, 40.714606]
        },
        "id": "6d726c5b-81a0-4a23-b010-43e8cb5c9163"
    }, {
        "type": "Feature",
        "properties": {
            "shape": "Marker",
            "name": "Unnamed Layer",
            "category": "default",
            "year": 2000,
            "type": "PUBLIC"
        },
        "geometry": {
            "type": "Point",
            "coordinates": [-73.995109, 40.715062]
        },
        "id": "0a354fb5-37e2-43cf-ab6d-71b4ded70d99"
    }, {
        "type": "Feature",
        "properties": {
            "shape": "Marker",
            "name": "Unnamed Layer",
            "category": "default",
            "year": 2000,
            "type": "COMMERCE"
        },
        "geometry": {
            "type": "Point",
            "coordinates": [-73.992877, 40.7216]
        },
        "id": "fd590e75-1098-447e-bccf-7ab618772a64"
    }]
};

// Ajout des données de l'objet GEOJSON à la carte
L.geoJSON(geojsonFeature).addTo(map);
```

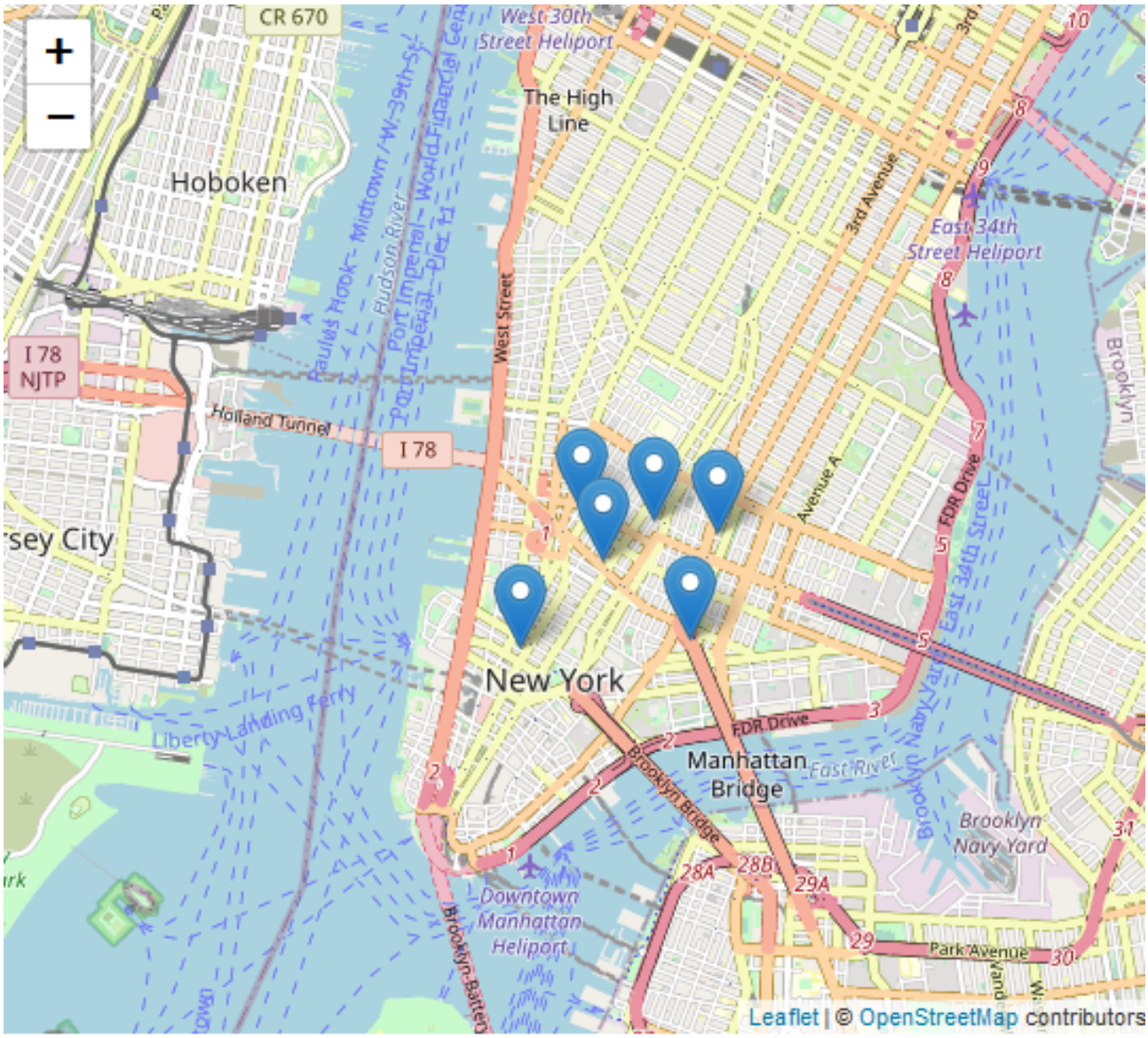


FIGURE 5.1. – On obtient une carte avec des points issus de l'objet GEOJSON

5.2. Créer l'interface

Maintenant qu'on a nos données on va créer l'interface permettant de les filtrer, voila à quoi elle doit ressembler:

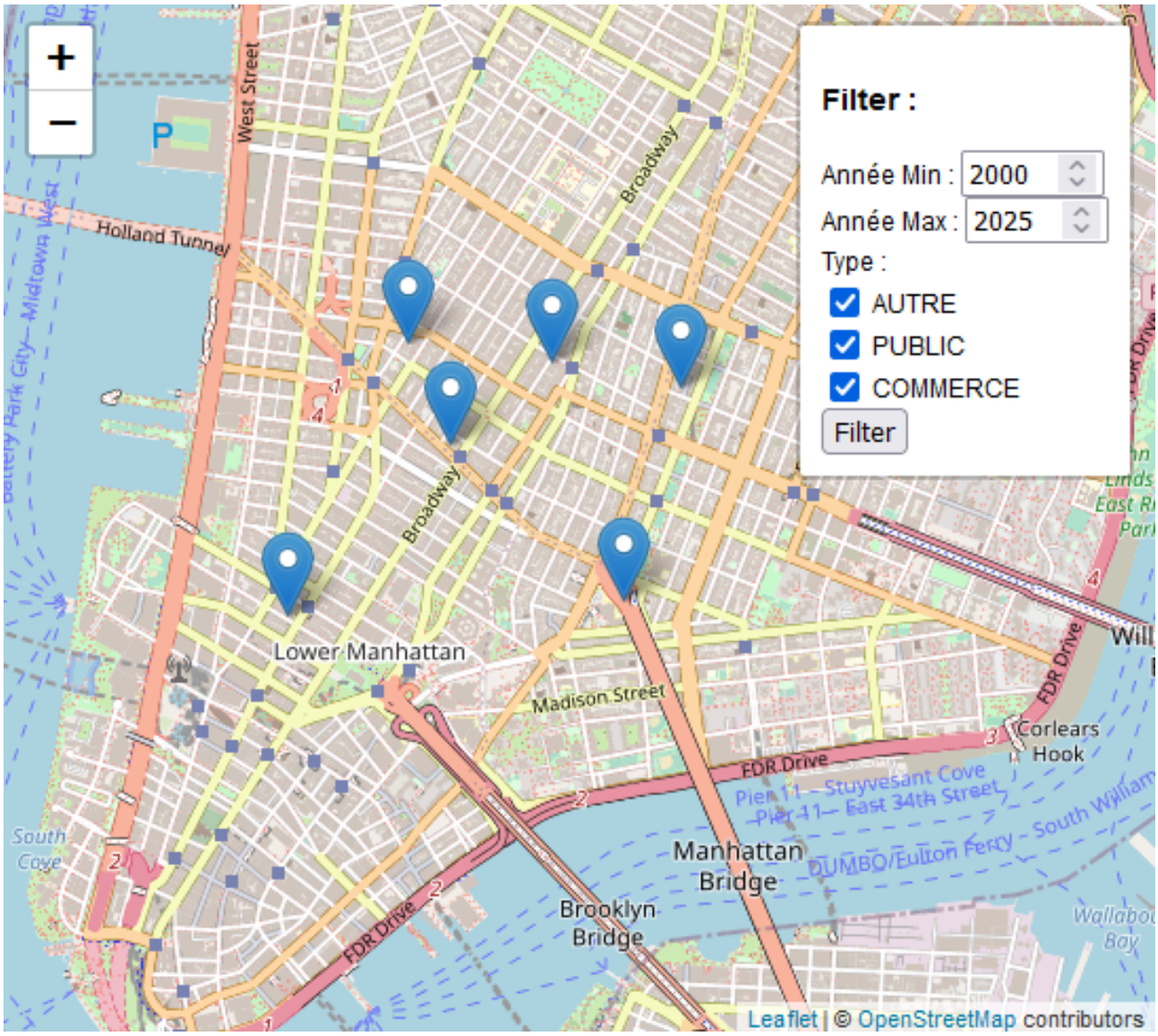


FIGURE 5.2. – Ce à quoi on veut que ressemble notre interface de filtrage

Pour cela, on reprend d'abord une interface vide comme dans le premier chapitre:

```
/*
 * Classe gérant l'interface utilisateur de filtrage
 */
let MyControlClass =  L.Control.extend({

  options: {
    position: 'topright'
  },

  /*
   * Ajout de l'interface à la carte
   */
  onAdd: function(map) {
```

```
14
15      this.map = map;
16
17      let div = L.DomUtil.create('div', 'leaflet-bar my-control');
18
19      return div;
20    },
21
22    /*
23     * Suppression de l'interface
24     */
25    onRemove: function(map)
26    {
27    }
28  });
29
30  // Ajout de l'interface utilisateur à la carte
31  let myControl = new MyControlClass().addTo(map);
```

Dans le méthode `onAdd`, on commence par la création du titre: (Création d'un élément HTML `h3` avec modification de son contenu HTML)

```
1     let title = L.DomUtil.create('h3', '', div);
2     title.innerHTML = "Filter : ";
```

Il nous faut des éléments permettant de filtrer les dates, pour cela on va ajouter deux `div` composés de:

— Un label
— Un champ input de type `number` avec une valeur par défaut

```
1     let divMin = L.DomUtil.create('div', '', div);
2     let labelMin = L.DomUtil.create('label', '', divMin);
3     labelMin.innerHTML = "Année Min : ";
4     let inputMin = L.DomUtil.create('input', 'input-number',
          divMin);
5     inputMin.type = "number";
6     inputMin.value = 2000;
```

Pour créer notre sélecteur de type, ont créé une div contenant des checkbox avec chacune un label:

```
1     let divType = L.DomUtil.create('div', '', div);
2     divType.innerHTML = "Type : ";
```

```
3
4       let divTypeOther = L.DomUtil.create('div', '', divType);
5       let inputOther = L.DomUtil.create('input', '', divTypeOther);
6       inputOther.type = "checkbox";
7       inputOther.checked = true;
8       let labelOther = L.DomUtil.create('label', '', divTypeOther);
9       labelOther.innerHTML = " AUTRE";
10
11      let divTypePublic = L.DomUtil.create('div', '', divType);
12      let inputPublic = L.DomUtil.create('input', '', divTypePublic);
13      inputPublic.type = "checkbox";
14      inputPublic.checked = true;
15      let labelPublic = L.DomUtil.create('label', '', divTypePublic);
16      labelPublic.innerHTML = " PUBLIC";
17
18      let divTypeBusiness = L.DomUtil.create('div', '', divType);
19      let inputBusiness = L.DomUtil.create('input', '',
            divTypeBusiness);
20      inputBusiness.type = "checkbox";
21      inputBusiness.checked = true;
22      let labelBusiness = L.DomUtil.create('label', '',
            divTypeBusiness);
23      labelBusiness.innerHTML = " COMMERCE";
24
25      var buttonFilter = L.DomUtil.create('button', '', div);
26      buttonFilter.innerHTML = "Filter";
```

Pour terminer, on gère l'évènement au clic qui appelle de la fonction `filter` qui effectuera le filtrage des données:

```
1   L.DomEvent.on(buttonFilter, 'click', function() {
        this.filter(parseInt(inputMin.value),
        parseInt(inputMax.value), inputOther.checked,
        inputPublic.checked, inputBusiness.checked); }, this);
```

Pour que le design soit plus agréable il faut également faire quelques modifications en CSS:

```
1  .my-control {
2    padding-left : 10px;
3    padding-right : 10px;
4    padding-bottom : 10px;
5    background-color : white;
6  }
7
8  .button-class {
9    padding : 2px;
```

```
  margin-top : 5px;
  width : 50px;
}

.input-number {
  width : 60px;
}
```

Code complet de l'interface:

```
  /*
   * Ajout de l'interface à la carte
   */
  onAdd: function(map) {

    this.map = map;

    let div = L.DomUtil.create('div', 'leaflet-bar my-control');

    let title = L.DomUtil.create('h3', '', div);
    title.innerHTML = "Filter : ";

                // Champ date (année) minimal
    let divMin = L.DomUtil.create('div', '', div);
    let labelMin = L.DomUtil.create('label', '', divMin);
    labelMin.innerHTML = "Année Min : ";
    let inputMin = L.DomUtil.create('input', 'input-number',
        divMin);
    inputMin.type = "number";
    inputMin.value = 2000;

                // Champ date (année) maximal
    let divMax = L.DomUtil.create('div', '', div);
    let labelMax = L.DomUtil.create('label', '', divMax);
    labelMax.innerHTML = "Année Max : ";
    let inputMax = L.DomUtil.create('input', 'input-number',
        divMax);
    inputMax.type = "number";
    inputMax.value = 2025;

                // Checkbox des types
    let divType = L.DomUtil.create('div', '', div);
    divType.innerHTML = "Type : ";

    let divTypeOther = L.DomUtil.create('div', '', divType);
    let inputOther = L.DomUtil.create('input', '', divTypeOther);
    inputOther.type = "checkbox";
    inputOther.checked = true;
```

```
37      let labelOther = L.DomUtil.create('label', '', divTypeOther);
38      labelOther.innerHTML = " AUTRE";
39
40      let divTypePublic = L.DomUtil.create('div', '', divType);
41      let inputPublic = L.DomUtil.create('input', '', divTypePublic);
42      inputPublic.type = "checkbox";
43      inputPublic.checked = true;
44      let labelPublic = L.DomUtil.create('label', '', divTypePublic);
45      labelPublic.innerHTML = " PUBLIC";
46
47      let divTypeBusiness = L.DomUtil.create('div', '', divType);
48      let inputBusiness = L.DomUtil.create('input', '',
            divTypeBusiness);
49      inputBusiness.type = "checkbox";
50      inputBusiness.checked = true;
51      let labelBusiness = L.DomUtil.create('label', '',
            divTypeBusiness);
52      labelBusiness.innerHTML = " COMMERCE";
53
54                  // Bouton de lancement de l'action de filtrage
55      var buttonFilter = L.DomUtil.create('button', 'button-class',
            div);
56      buttonFilter.innerHTML = "Filter";
57
58      L.DomEvent.on(buttonFilter, 'click', function() {
            this.filter(parseInt(inputMin.value),
            parseInt(inputMax.value), inputOther.checked,
            inputPublic.checked, inputBusiness.checked); }, this);
59
60      return div;
61  },
```

5.3. Filtrer les données

On comment par créer la structure de notre fonction de filtrage:

```
1   /*
2    * Filtrage les données du GEOJSON
3    * @param {number}          startDate                 L'année
      de début
4    * @param {number}          endDate                   L'année
      de fin
5    * @param {boolean}         inputOtherChecked         L'état
      "coché" de l'input "AUTRE"
6    * @param {boolean}         inputPublicChecked        L'état
      "coché" de l'input "PUBLIC"
```

```
7     * @param {boolean}         inputBusinessChecked       L'état
       "coché" de l'input "COMMERCE"
8     */
9    filter(startDate, endDate, inputOtherChecked,
         inputPublicChecked, inputBusinessChecked) {
10
11   },
```

Pour filtrer nos données, il nous faut supprimer nos layers actuellement affichés puis recharger nos données avec la fonction `filter` qui va nous permettre de sélectionner les données que l'on souhaite récupérer:

```
1      // Retrait des layers de la carte (données issues du GEOJSON)
2      map.removeLayer(layers);
3
4      // Rechargement des données du GEOJSON
5      layers = L.geoJSON(geojsonFeature,
6      {
7          filter: function (feature) {
8
9              // Filtre des données : si return false la donnée
                   n'est pas récupérée
10
11             return true;
12         }
13     }).addTo(map);
```

On crée des filtres sur les dates, si des dates ont été remplies dans le formulaire on rejette les éléments dont l'année n'est pas comprise entre la date de début et la date de fin:

```
1  if (startDate && endDate) {
2      if (endDate) {
3          if (!(feature.properties.year >= startDate &&
               feature.properties.year <= endDate))
4              return false;
5      } else if (feature.properties.year < startDate)
6          return false;
7  }
```

Enfin on filtre les types, si la case correspondant au type de l'élément n'est cochée, on ne récupère pas cet élément:

```
if(feature.properties.type == "AUTRE" && !inputOtherChecked) {
 return false;
}
else if(feature.properties.type == "PUBLIC" &&
    !inputPublicChecked) {
 return false;
}
else if(feature.properties.type == "COMMERCE" &&
    !inputBusinessChecked) {
 return false;
}
```

Code des filtres complet:

```
  /*
   * Filtrage les données du GEOJSON
   * @param {number}          startDate                 L'année
    de début
   * @param {number}          endDate                   L'année
    de fin
   * @param {boolean}         inputOtherChecked         L'état
    "coché" de l'input "AUTRE"
   * @param {boolean}         inputPublicChecked        L'état
    "coché" de l'input "PUBLIC"
   * @param {boolean}         inputBusinessChecked      L'état
    "coché" de l'input "COMMERCE"
   */
  filter(startDate, endDate, inputOtherChecked,
      inputPublicChecked, inputBusinessChecked) {
    // Retrait des layers de la carte (données issues du GEOJSON)
    map.removeLayer(layers);

    // Rechargement des données du GEOJSON
    layers = L.geoJSON(geojsonFeature,
    {
        filter: function (feature) {
            // Filtre des dates
            if (startDate && endDate) {
                if (endDate) {
                    if (!(feature.properties.year >= startDate &&
                        feature.properties.year <= endDate))
                        return false;
                } else if (feature.properties.year < startDate)
                    return false;
            }

            // Filtre des types
```

```
27              if(feature.properties.type == "AUTRE" &&
                    !inputOtherChecked) {
28               return false;
29              }
30              else if(feature.properties.type == "PUBLIC" &&
                    !inputPublicChecked) {
31               return false;
32              }
33              else if(feature.properties.type == "COMMERCE" &&
                    !inputBusinessChecked) {
34               return false;
35              }
36
37              return true;
38          }
39      }).addTo(map);
40  },
```

5.4. Résultat et code complet

Voici ce le résultat, nous avons notre interface de filtrage fonctionnelle:

!(https://jsfiddle.net/dqgtnku5/5/)

Conclusion

Vous savez maintenant comment filtrer des données, le filtrage de données de type `Polygon` fonctionne de la même manière. Il est également possible de filtrer des données selon d'autres critères notamment leur géométrie pour par exemple exclure des points situés dans une zone cible.

Conclusion

Je vous ai présenté les éléments les plus importants à mes yeux de Leaflet mais il en existe plein d'autres notamment à travers les très nombreux plugins créés par la communauté.

J'espère que cela vous aidera dans vos projets et que cela donnera lieu à de superbes cartes.
🍊