



Adrien PAVIE

Contributeur OpenStreetMap et entrepreneur en géomatique, je développe des solutions d'analyse et de visualisation de données géographiques pour mieux comprendre le monde qui nous entoure.
<https://pavie.info/>

niveau
100

Partie 2

OpenStreetMap : afficher la carte avec Leaflet et Mapbox GL !

Dans l'article précédent, nous avons vu ensemble l'intérêt d'utiliser les données géographiques issues d'OpenStreetMap. L'usage premier, qui vient naturellement à l'esprit, est de pouvoir en réaliser une carte, support graphique facilement lisible pour représenter l'espace ou contextualiser une information. Traditionnellement, ces cartes existaient sous forme papier (cartes routières, plans d'Etat-major, représentations topographiques...) et avec l'émergence des systèmes informatiques, elles ont pris forme numérique dans nos boîtiers GPS, sur le web ou sur nos smartphones. Cet article va s'intéresser aux méthodes et outils que vous pourrez utiliser pour afficher une carte interactive sur vos sites web.

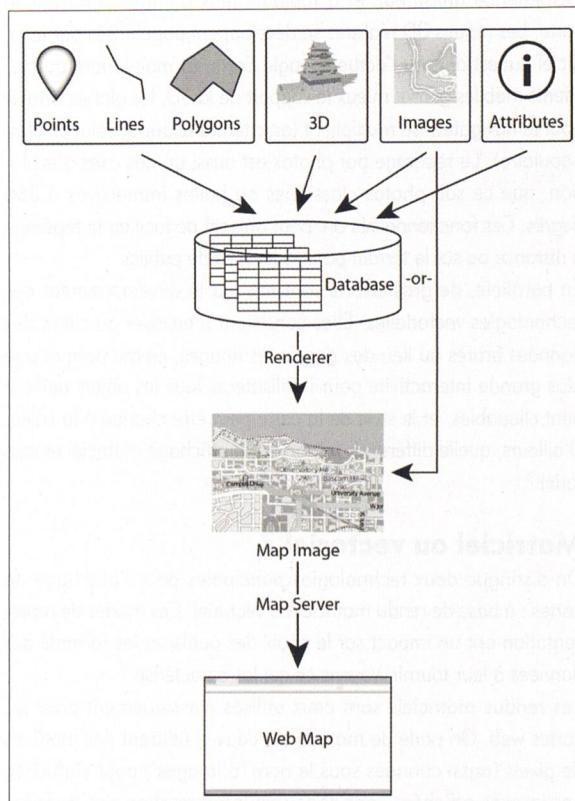
C'est un sujet dans l'air du temps, suite au changement récent de la politique tarifaire de Google Maps qui, pendant de nombreuses années, a proposé ses services cartographiques à prix concurrentiels. Depuis cet été, les conditions ont changé, entraînant une hausse des tarifs et une obligation de saisir des coordonnées bancaires. Les utilisateurs pieds et poings liés doivent donc soit passer à la caisse, soit changer de fournisseur, soit voir leur carte inutilisable. L'impact de ce changement est déjà visible, de nombreux sites web affichant une erreur au chargement de la carte de Google.

Comment fonctionne une carte web ? Quelles technologies existent pour gérer leur affichage ? Quelles sont les possibilités de personnalisations ? Nous allons le découvrir en nous intéressant à la théorie derrière les cartes web, puis en la mettant en pratique avec les bibliothèques Leaflet et Mapbox GL.

Les cartes pour le web

L'histoire des cartes ne date pas d'hier, mais celle des cartes pour le web est largement plus récente. Avec la naissance du World Wide Web en 1989, la mise à disposition de ressources documentaires à l'échelle mondiale est devenue possible. Les premiers pas des cartes interactives ont eu lieu en 1993, lorsque le centre de recherche Xerox (Xerox PARC) a lancé un service nommé « Map Viewer », dont l'objet était d'afficher une carte dont on pouvait modifier l'apparence et la zone affichée à l'écran. Cette application a posé les bases des cartes sur le web, mais leur évolution a continué. Comme dans de nombreux domaines de l'informatique, ces technologies sont passées par l'étape de la normalisation. L'Open Geospatial Consortium est un consortium mondial qui définit les normes autour des solutions géospatiales. Cet organisme est à l'origine de la norme « Web Map Service » (WMS), initiée en 1999, dont l'objectif est de formaliser les appels aux serveurs cartographiques pour la mise à disposition de cartes sur le web. On citera également le standard « Web Feature Service », similaire à WMS, mais pour la mise à disposition de données brutes. Ces standards ont facilité l'interopérabilité des logiciels sur le marché, et la communication entre les briques serveurs et clients.

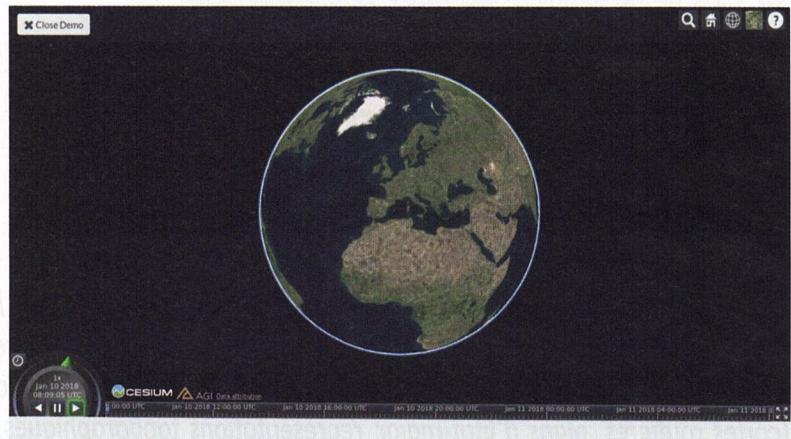
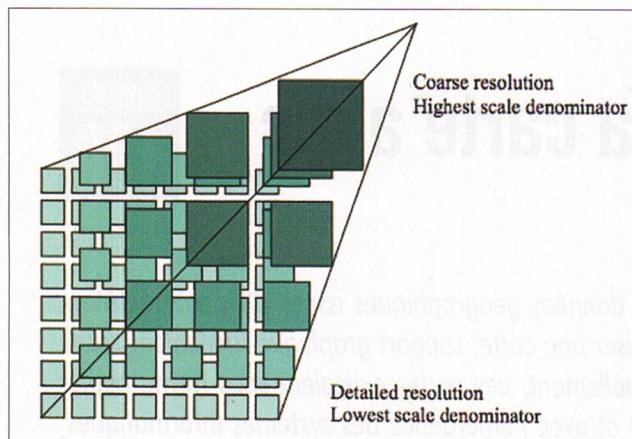
Plusieurs innovations technologiques ont permis d'accélérer l'affichage et faciliter la navigation de l'utilisateur. Une innovation



Mise à disposition de cartes sur le web

majeure consiste à gérer l'affichage non pas sous forme d'une seule image renvoyée à l'utilisateur, mais à l'aide d'une grille de tuiles. Cette technique, popularisée par Google Maps (sorti en 2005), consiste à découper la surface de la terre en une grille à plusieurs niveaux, où chaque carreau correspond à une petite image carrée. De cette manière, il devient possible de créer un déplacement fluide de la carte, en permettant un chargement progressif des images de fond. La plupart des technologies actuelles pour la représentation cartographique sur le web reposent toujours sur ce principe.

L'évolution des cartes sur le web n'est pas terminée. Le futur nous réserve encore des surprises. La tendance est à l'amélioration de



l’expérience utilisateur, et à toujours plus d’immersion dans la carte. Les rendus 3D réalisistes se développent, popularisés par le logiciel qui est devenu l’actuel Google Earth. Et maintenant que les clients mobiles gèrent mieux le support de la 3D, les globes virtuels pour le navigateur se multiplient (on citera CesiumJS, solution libre populaire). Le repérage par photos est aussi un des axes d’évolution, que ce soit photos classiques ou bulles immersives à 360 degrés. Ces fonctionnalités ont pour objectif de faciliter le repérage à distance ou sur le terrain pour tous types de publics. En parallèle, de gros efforts sont mis sur le développement des technologies vectorielles. Elles consistent à envoyer au client des données brutes au lieu des classiques images, ce qui permet une plus grande interactivité pour l’utilisateur. Tous les objets deviennent cliquables, et le style de la carte peut être changé à la volée. D’ailleurs, quelle différence fait-on entre affichage matriciel et vectoriel ?

Matriciel ou vectoriel ?

On distingue deux technologies principales pour l’affichage de cartes : à base de rendu matriciel ou vectoriel. Ces modes de représentation ont un impact sur le choix des outils, et les formats des données à leur fournir. Voyons ce qui les caractérise.

Les rendus matriciels sont ceux utilisés classiquement pour les cartes web. On parle de matrice car ceux-ci utilisent des matrices de pixels (aussi connues sous le nom "d’images") pour s’afficher. Les images affichées sont donc une interprétation des données brutes, avec un style particulier, décidé par le fournisseur de la carte. Ils sont présents historiquement de par la « simplicité » de la technologie : générer, mettre à disposition, et afficher un ensemble d’images sont des fonctionnalités bien gérées par les machines et logiciels depuis plusieurs décennies.

Les rendus vectoriels ont été popularisés plus récemment. On parle de vecteurs car les données sont mises à disposition sous forme brute, avec une description à base de géométries auxquelles sont associés des attributs. Ces représentations se basent donc sur des formats très proches de ceux utilisés pour les données OpenStreetMap, et plus largement les données géographiques en général. Leur usage est plus récent pour l’affichage de cartes du fait de certaines contraintes techniques, notamment sur la capacité des machines clientes à les analyser pour affichage dans le navigateur.

Ces deux technologies présentent respectivement des avantages et inconvénients, qu’il est bon d’avoir en tête lorsque l’on travaille à l’affichage de cartes web.

Technologie	Matriciel	Vectoriel
Application du style	Côté serveur	Côté serveur ou client
Charge pour la création (serveur)	Forte	Faible
Bande passante nécessaire	Moyenne	Faible
Charge pour le rendu (client)	Faible	Forte
Bibliothèques disponibles	NOMBREUSES ET éprouvées	Peu nombreuses et plus jeunes
Personnalisation côté client	Très limitée et coûteuse	LARGE ET PEU COÛTEUSE
Efforts de développement pour le futur	Limités	Nombreux

Les cas d’usages pour chaque technologie sont différents, on préférera utiliser le système matriciel pour de l’affichage mobile avec une connexion Internet plutôt rapide, mais on favorisera le système vectoriel pour de l’affichage hors-ligne ou sur des applications avec modification du style par l’utilisateur à la volée.

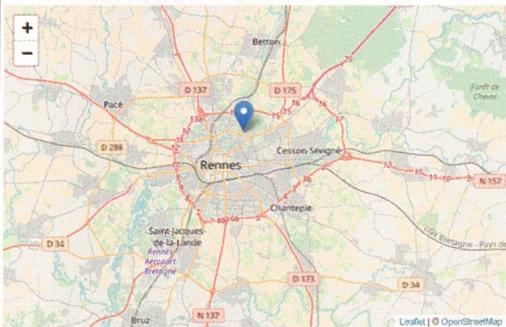
Maintenant que nous y voyons plus clair sur ces deux modes de gestions des données, nous allons découvrir une bibliothèque JavaScript pour chacun des systèmes : Leaflet pour l’affichage matriciel, et Mapbox GL pour l’affichage vectoriel. Dans les deux cas, on cherchera à montrer le fond de carte basé sur les données OpenStreetMap et quelques possibilités d’interactions.

Carte simple avec Leaflet



Leaflet est une bibliothèque JavaScript, libre de droits (licence BSD), qui gère l’affichage de cartes interactives pour le web. Son développement a débuté en 2011, et la version actuelle est la 1.3. Elle est utilisée par de très nombreux acteurs, que ce soit Wikipédia, l’IGN, le Wall Street Journal... Comme de nombreuses bibliothèques cartographiques, Leaflet gère un ensemble de calques, que ce soient des fonds de carte ou des données en surcouche. Elle

Une carte sympa



Votre carte interactive avec Leaflet et OpenStreetMap

permet d'utiliser des données dans des formats très variés, que ce soit nativement ou à l'aide d'extensions : cartes TMS (grille de tuiles) ou WMS (carte par emprise), données GeoJSON, images brutes, vidéos... Par ailleurs, Leaflet peut être utilisé pour afficher plus que des cartes : il est possible de configurer toutes sortes d'images découpées par tuiles. On peut ainsi afficher des cartes de mondes de jeux vidéo ou des photos à très hautes résolutions.

L'objectif ici va être d'afficher un fond de carte utilisant les données OpenStreetMap, et de venir ajouter en surcouche un marqueur interactif. Pour cela, on va débuter par le chargement de la bibliothèque. Créez une page HTML à la structure classique, et y ajoutez-y les appels aux scripts Leaflet.

```
<link rel="stylesheet" href="https://unpkg.com/leaflet@1.3.4/dist/leaflet.css" />
<script src="https://unpkg.com/leaflet@1.3.4/dist/leaflet.js"></script>
```

Une fois fait, nous devons déclarer le bloc réservé à la carte. Créez une balise div dans votre page en précisant un identifiant.

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

Et assurez-vous d'avoir bien défini une hauteur à la carte, sans quoi celle-ci ne s'affichera pas. Nous allons le faire en CSS.

```
#mapid { height: 180px; }
```

Nous avons désormais tous les éléments préparés, passons à linitialisation de la carte, qui va être réalisée en JavaScript. Lensemble des fonctions de Leaflet sont mises à disposition à travers la variable globale L. La première fonction que nous allons appeler est L.map, qui permet de créer la carte vide dans le bloc « mapid » que nous avons déclaré précédemment.

```
var laCarte = L.map('mapid');
```

Si vous chargez votre page, vous devez voir le bloc apparaître avec les boutons de zoom, mais rien de plus. Eh oui, il nous reste à préciser ce que l'on souhaite afficher ! Pour cela, on commence par choisir la zone initiale avec setView, à appeler sur l'objet laCarte, et qui prend comme paramètres latitude, longitude, et un niveau de zoom (de 1 à 19, 19 étant le plus grand).

```
laCarte.setView([48.11, -1.65], 11);
```

Nous allons maintenant choisir quel fond de carte Leaflet doit afficher. Il existe de nombreux rendus de données géographiques disponibles sur le web. Ici, nous allons prendre le rendu généraliste proposé par OpenStreetMap. Pour créer un fond de carte à partir d'un service d'images tuilées, on utilise l'objet L.tileLayer.

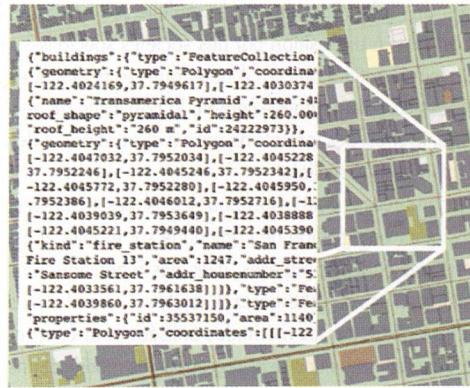


Illustration du contenu d'une tuile vectorielle

```
var leFondOpenStreetMap = L.tileLayer('https://s.tile.openstreetmap.org/{z}/{x}/{y}.png', {
  maxZoom: 19,
  attribution: '&copy; <a href="http://www.openstreetmap.org/copyright">OpenStreetMap</a>'
});
```

Il est essentiel de toujours faire apparaître les crédits du fond de carte, obligatoires du fait de la licence utilisée. Le fond est créé, mais il reste à l'ajouter à la carte en elle-même. Pour cela, on utilise la fonction addTo sur l'objet en lui-même.

```
leFondOpenStreetMap.addTo(laCarte);
```

Désormais, si vous chargez la page, vous verrez le fond OpenStreetMap s'afficher. Bien joué ! Mais allons plus loin, ajoutons maintenant un marqueur interactif par-dessus la carte, qui affichera une infobulle au clic utilisateur. Leaflet embarque un objet L.marker prêt-à-l'emploi. On peut ainsi lui passer en paramètres les coordonnées latitude/longitude où celui-ci doit apparaître. De la même manière que pour l'objet L.tileLayer, il est nécessaire d'ajouter le marqueur à la carte après sa création avec la fonction addTo.

```
var leMarqueur1 = L.marker([48.13, -1.66]);
leMarqueur1.addTo(laCarte);
```

D'autres options de personnalisation sont disponibles : changer l'icône, ajouter un libellé au survol, gestion des événements utilisateurs... Ici, nous allons afficher une infobulle. Celle-ci peut prendre du contenu au format HTML.

```
leMarqueur1.bindPopup('<h3>Un endroit sympa</h3>');
```

Si vous chargez la page, vous verrez donc le fond de carte ainsi que ce marqueur. Vous voyez qu'il est assez simple, en quelques lignes de code, d'embarquer votre propre carte interactive sur un site web.

Le fond de carte OpenStreetMap est utilisable assez simplement. Il est à noter que, comme la plupart des projets collaboratifs, l'infrastructure est maintenue de manière associative, financée par les dons des adhérents ou partenaires. Ainsi ces tuiles sont utilisables librement pour un usage raisonnable (volume de requêtes faible). Si vous souhaitez faire un usage plus conséquent, il est indispensable

de s'appuyer sur une infrastructure dédiée, qu'elle soit hébergée par vos soins, ou mise à disposition par des sociétés spécialisées (voir « Aller plus loin » en fin d'article). À noter que ces prestataires offrent des services complémentaires comme la possibilité de personnaliser le style, le choix des formats de mise à disposition, et des tarifs adaptés selon l'usage réalisé.

Leaflet offre de nombreuses possibilités pour les usages classiques, et permet nativement une gestion de l'affichage sur mobile. Il est possible de créer des cartes complexes, avec des représentations avancées. Toutefois, plus vous chargerez d'objets sur la carte, plus vous aurez à être vigilant sur les méthodes utilisées pour charger des données (passer en mode canevas, regrouper les modifications du DOM, opter pour des représentations sous forme de clusters...). Pour certains usages, soit orientés SIG (systèmes d'information géographique), soit avec une vaste gamme d'interactions avec l'utilisateur, on préférera s'orienter vers d'autres technologies. Voyons tout de suite l'une d'entre elles : Mapbox GL.

Carte personnalisable avec Mapbox GL

Mapbox GL est le pendant vectoriel de Leaflet. Cette solution libre de droits (sous licence BSD) est développée par la société établienne Mapbox, spécialisée dans les services autour des données OpenStreetMap. Cet outil permet d'afficher des cartes basées sur les technologies vectorielles. Il existe sous différentes déclinaisons selon les plateformes web et mobiles. La version JavaScript a été lancée en 2014. Avant de nous lancer dans la mise en place de la carte, nous devons nous assurer de pouvoir accéder à une source de tuiles vectorielles, c'est-à-dire les données qui seront affichées en fond de carte. Plusieurs fournisseurs existent, Mapbox propose ses propres tuiles, mais nous utiliserons ici un fournisseur nommé Jawg Maps, basé à Paris. Je vous invite à vous créer un compte à cette adresse, et de noter la clé d'API qui vous sera fournie, nous en aurons besoin pour la suite : <https://www.jawg.io/lab/>

De manière similaire à ce que vous avez fait pour Leaflet, on va utiliser une structure de page HTML basique, puis importer les scripts de la bibliothèque.

```
<script src='https://api.tiles.mapbox.com/mapbox-gl-js/v0.48.0/mapbox-gl.js'></script>
<link href='https://api.tiles.mapbox.com/mapbox-gl-js/v0.48.0/mapbox-gl.css' rel='stylesheet' />
```

Créez un bloc div pour afficher la carte.

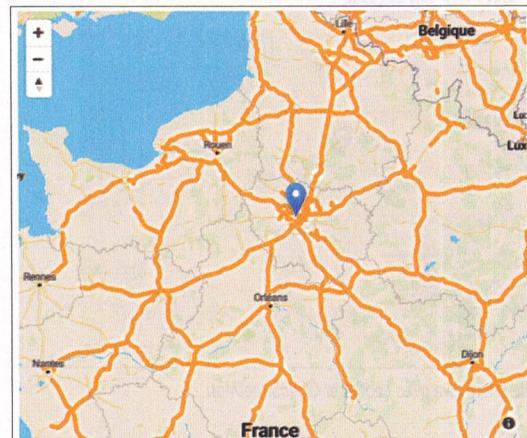
```
<div id='map'></div>
```

Puis, on définira les dimensions de ce bloc à l'aide de styles CSS.

```
#map { width: 400px; height: 300px; }
```

Votre structure de page est prête, nous allons pouvoir passer aux choses sérieuses ! Comme vu pour Leaflet, il est nécessaire d'initialiser la bibliothèque Mapbox GL en lui précisant dans quel bloc elle doit construire la carte. On précisera également le fond à utiliser, et les coordonnées de départ.

```
var CLE_API = "votre clé d'API";
var map = new mapboxgl.Map({
    container: 'map',
    style: "https://tile.jawg.io/jawg-dark.json?access-token=" + CLE_API,
```



Votre carte interactive avec Mapbox GL et OpenStreetMap

```
zoom: 2,
center: [2.3210938, 48.7965913]
});
```

Pour que le chargement des données se fasse correctement et sans erreurs de sécurité, il est nécessaire de servir la page à travers un serveur web. Vous pouvez lancer un serveur web minimaliste avec cette commande Python 3 dans le dossier contenant votre fichier HTML, celui-ci sera alors disponible à l'adresse <http://localhost:3000/>

```
python -m http.server 3000
```

A ce stade, vous devez voir la carte apparaître. Par défaut, celle-ci n'affiche pas de contrôles pour gérer le zoom et l'attribution de la carte. Vous pouvez les ajouter en utilisant les objets NavigationControl et AttributionControl. D'autres widgets de ce type sont disponibles (affichage d'une échelle, d'un bouton de géolocalisation automatique ou passage en plein écran).

```
map.addControl(new mapboxgl.NavigationControl(), 'top-left');
map.addControl(new mapboxgl.AttributionControl({ customAttribution: 'Ma super
carte, données &copy; OpenStreetMap' }));
```

Vous l'aurez compris, l'intérêt d'utiliser des données vectorielles est de pouvoir modifier le rendu des objets dynamiquement, côté client. Cela est possible en changeant les propriétés de style embarquées par défaut dans les tuiles que l'on récupère auprès du fournisseur. C'est le rôle de la fonction setPaintProperty. Celle-ci prend en paramètre un nom de couche, la propriété à modifier et sa nouvelle valeur. Exemple :

```
map.setPaintProperty('building', 'fill-color', 'red');
```

Les couches disponibles dépendent des tuiles que vous récupérez. Pour afficher la liste, on utilisera le code suivant :

```
console.log(Object.keys(map.style._layers));
```

Il est à noter que les modifications de style ne peuvent s'effectuer qu'une fois la carte chargée et un premier rendu réalisé. Pour ce faire, on doit donc écouter l'événement de chargement de la carte, et réaliser la modification de style en réaction à cet événement. De manière classique dans les bibliothèques JavaScript, on dispose d'une méthode « on » pour réagir aux événements lancés. Si l'on remet tous les éléments en place, on pourra donc modifier notre style avec le code suivant :

```
map.on('load', function() {
    console.log(Object.keys(map.style._layers));
    map.setPaintProperty('building', 'fill-color', 'red');
    map.setPaintProperty('road-motorway', 'line-width', 5);
});
```

Vous devez donc voir apparaître les autoroutes en surépaisseur, et les bâtiments en rouge. Cette méthode d'édition de style est utile lorsque l'on cherche à réaliser des modifications mineures. Pour créer un style complètement différent, on préférera créer un modèle directement auprès du fournisseur de tuiles.

Maintenant que vous avez la main sur la carte et son style, nous allons ajouter un marqueur interactif. Cette opération est à réaliser en plusieurs étapes :

- Chargement du symbole pour le marqueur avec `map.loadImage` ;
- Ajout du symbole dans les données disponibles avec `map.addImage` ;
- Ajout du marqueur avec ce symbole à la carte avec `map.addLayer`.

Ce qui nous donne le code suivant :

```
map.loadImage('https://raw.githubusercontent.com/Leaflet/Leaflet/master/dist/images/marker-icon.png', function(error, image) {
    if (error) throw error;
    map.addImage('marker', image);
    map.addLayer({
        "id": "points",
        "type": "symbol",
        "source": {
            "type": "geojson",
            "data": {
                "type": "FeatureCollection",
                "features": [
                    {
                        "type": "Feature",
                        "geometry": {
                            "type": "Point",
                            "coordinates": [2.3, 48.8]
                        }
                    }
                ]
            }
        },
        "layout": {
            "icon-image": "marker",
            "icon-size": 1,
            "icon-anchor": "bottom"
        }
    });
});
```

La fonction `map.addLayer` prend en paramètre une configuration assez dense, car cette méthode est en mesure de créer tout type de couche pour Mapbox GL. On retrouve principalement le type de couche (ici « symbol »), la source des données (ici notre point) et des options d'affichage (propriétés « icon-* »). Si vous rechargez la page, vous verrez le marqueur apparaître, mais il n'offre pas encore de possibilités d'interactivité. Pour ce faire, on va spécifier une fonction de réaction à l'événement au clic sur le calque qui contient notre marqueur. Elle déclenchera la création d'un objet de type

Popup avec le texte de notre choix.

```
map.on('click', 'points', function (e) {
    var coordinates = e.features[0].geometry.coordinates.slice();

    // Correction des coordonnées lorsque la carte est largement dézoomée
    while (Math.abs(e.lngLat.lng - coordinates[0]) > 180) {
        coordinates[0] += e.lngLat.lng > coordinates[0] ? 360 : -360;
    }

    new mapboxgl.Popup()
        .setLngLat(coordinates)
        .setHTML("Ceci est un point")
        .addTo(map);
});
```

Une correction des coordonnées est réalisée dans le cas où la carte est à un niveau de zoom très large, ce qui rend possible l'affichage du même marqueur à plusieurs endroits de la carte, comme celle-ci se répète. Voilà, vous avez désormais votre propre carte web, à la fois interactive et personnalisable ! Mapbox GL offre de nouvelles perspectives pour les cartes interactives sur le web. Basé sur le moteur WebGL, l'exécution des calculs pour le rendu affiche une performance respectable. Vous l'aurez remarqué, ses appels sont relativement verbeux (à fonctionnalités équivalentes les appels sont environ deux fois plus longs qu'avec Leaflet). Cela est dû à la jeunesse de cette bibliothèque, on imagine qu'à l'avenir une simplification de ces appels sera réalisée pour disposer d'une meilleure lisibilité de code. Son développement est particulièrement actif, de nouvelles fonctionnalités émergeront à l'avenir. On pourra notamment espérer un support plus large de formats de données géographiques, d'enormes meilleures performances, et la gestion d'objets 3D complexes.

Conclusion

Nous avons découvert ensemble le monde des visualiseurs cartographiques pour le web. Vous savez désormais quelle est l'origine de ces systèmes, leur fonctionnement, et la différence entre mode vectoriel ou matriciel. Nous avons vu que les usages de ces deux modes sont différents, et que ces technologies sont donc complémentaires. Et vous avez mis en pratique l'utilisation basique des bibliothèques emblématiques Leaflet et Mapbox GL, ce qui vous a permis de découvrir les possibles pour vos applications web. J'espère que cet article d'initiation vous a donné envie d'en découvrir davantage et d'explorer les nombreuses possibilités offertes pour vos projets !

Pour aller plus loin

- Changement de tarifs de Google Maps :
<https://medium.com/@cq94/95f2e8dfaad>
- Documentation de Leaflet : <https://leafletjs.com/>
- Fonds de cartes de démo pour Leaflet :
<https://leaflet-extras.github.io/leaflet-providers/preview/>
- Sociétés fournissant des fonds de carte :
<https://switch2osm.org/fr/prestataires/>
- Documentation de Mapbox GL :
<https://www.mapbox.com/mapbox-gl-js/>
- Liste des propriétés de style disponibles pour Mapbox GL :
<https://www.mapbox.com/mapbox-gl-js/style-spec>



Jordan NOURRY
Développeur
@La combe du lion vert



Fouad JADOUANI
Développeur
@Société Générale

Refactoring de Legacy Code avec Programmation Fonctionnelle en pur JavaScript

Partie 2

La programmation fonctionnelle est devenue un sujet très tendance ces dernières années. Complètement méconnue des développeurs débutants, pour la plupart concentrés sur la programmation orientée objet, elle permet de rendre le code plus concis, plus simple et plus expressif.

Comment appréhender ce style de programmation ? Quels sont ses points forts ? Et surtout comment « refactorer » du code legacy en y ajoutant de la programmation fonctionnelle ? Pour étayer nos propos nous allons nous appuyer sur le langage JavaScript, dont la communauté a énormément travaillé ces 3 dernières années à l'intégration de la programmation fonctionnelle dans notre environnement professionnel !

niveau
200



LA CURRYIFICATION

Dans l'exemple précédent (n°222), on aurait également pu écrire la fonction `addArticlesToAuthor` de cette manière :

```
const addArticlesToAuthor = (author, articles) => (...author, articles);
```

Mais nous avons préféré l'écrire sous sa forme curryfiée.

La **curryification**⁽⁹⁾ est la transformation d'une fonction à plusieurs arguments en une fonction à un seul argument qui retourne une fonction sur le reste des arguments. Cela dans l'objectif de créer des **fonctions pures**. Ce qui nous donne une forme de réutilisabilité de code pour de nouvelles combinaisons et compositions.

Ceci introduit directement l'un des concepts qui justifie de fixer l'ensemble de ces règles lorsque nous faisons de la programmation fonctionnelle : l'**application partielle**.

L'APPLICATION PARTIELLE

La curryification permet l'application partielle d'une fonction. Vous comprendrez mieux avec un exemple :

```
const mult = a => b => a * b;

const identity = mult(1);
const double = mult(2);
const triple = mult(3);

[identity, double, triple]
  .map(fun => fun(42))
  .forEach(value => console.log(value));
```

Dans l'exemple précédent nous retrouvons la fonction `mult` qui a été curryfiée, que nous spécialisons en trois différentes fonctions `identity`, `double` et `triple`. Chacune fait une application partielle de la fonction '`mult`', pour définir une fonction qui multiplie par 1, 2 ou 3 le 2ème argument qui sera donné plus tard. Puis nous pouvons itérer sur les fonctions et leur passer le 2ème argument attendu pour réaliser le calcul, puis les afficher à la console.

(9) <https://fr.wikipedia.org/wiki/Curryfication>

L'application partielle est une arme redoutable quand il s'agit de faire un « refactoring » de fonction procédurale qui a beaucoup d'arguments. Car nous pouvons ainsi curryfier cette fonction et appliquer partiellement les arguments au plus tôt, ce qui permet d'avoir une fonction réalisant uniquement le calcul demandé.

RECURSION

En programmation fonctionnelle, la récursion est préférée aux systèmes de boucles traditionnels `while` ou `for`. Cela a pour avantage de maintenir un système sans état et de ne pas avoir d'effets de bords. Par exemple pour calculer le factoriel d'une valeur `n` en utilisant un état interne, nous aurions le code suivant :

```
const factorial = n => {
  let result = 1;
  for(let x = 1; x <= n; x+=1){
    result = x * result;
  }
  return result;
}
```

Voici la version sans état ...

```
const recursiveFactorial = n => {
  if(n < 2) {
    return n;
  }
  return n * recursiveFactorial(n - 1);
}
recursiveFactorial(5);
```

Quant au `ForEach`, il est aussi à éviter, car quand on regarde de plus près sa signature, on peut constater qu'il ne retourne pas de valeur. `ForEach` ne respecte donc pas la transparence référentielle et est donc propice aux effets de bord.

RECHERCHE ET TRANSFORMATION

Pour ce qui est des recherches dans une liste ou encore des transformations d'éléments d'un tableau, il faut privilégier les fonctions