

---

Mai 2022

Rapport de projet

# **HACKATHON**

## **DGFIP**

# TRAVAIL PRÉLIMINAIRE

## PROBLÉMATIQUE

La **couverture** du territoire par les structures DGFIP est-elle **optimale** et assure-t-elle un **égal accès** de tous au service public ?

## EXPLICATION DE LA DÉMARCHE

A l'étude de la problématique nous avons relevé plusieurs termes importants:

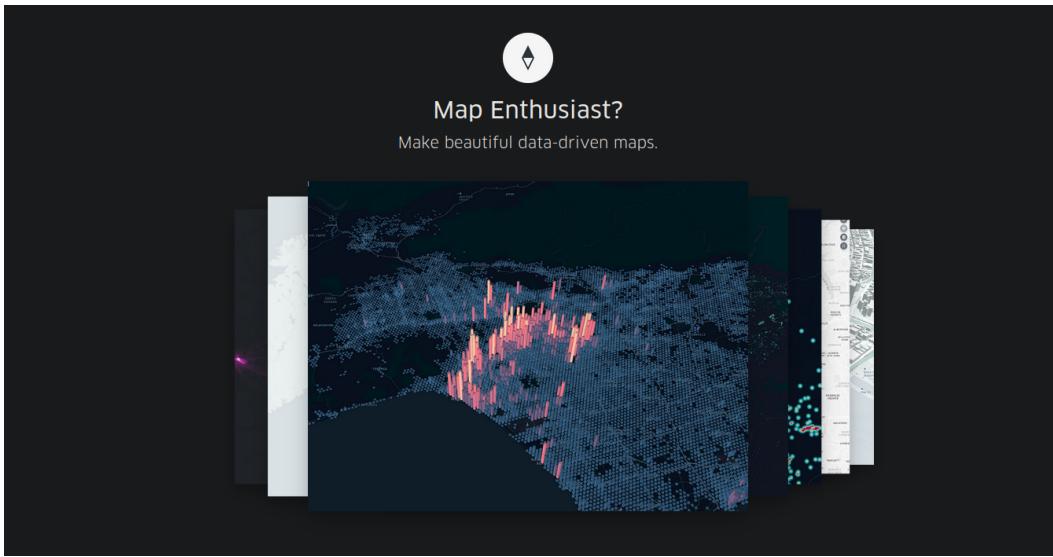
**"Couverture"** | **"Territoire"** = notion de **spatialité**.  
**"Optimale ?"** | **"égale ?"** = notion de **disparité**.

Nous avons donc orienté notre travail vers cette notion de mise en image des disparités sur le territoire sous forme d'une **carte 3D**. Notre but a été de créer un outil de visualisation pour la DGFIP qui soit **visible, fonctionnel** et **simple** à comprendre et à mettre à jour.

## RECHERCHES DOCUMENTAIRES PRÉLIMINAIRES

### Un outil de visualisation en 3D : Kepler.GL

**Kepler.gl** est une librairie open source développée par la société Uber, et spécialisée dans l'affichage de large jeux de données sur des **maps 3D**. La librairie est basée sur la librairie de modélisation 3D Decke.gl et développée sur en Reactjs en combo avec redux.js.



Grâce à un système d'inputs, elle permet de facilement intégrer de larges jeux de données sur des maps **personnalisables** sans même avoir à passer par le code, ce qui la rend extrêmement **accessible**. Elle propose aussi plusieurs options permettant de **manipuler** l'affichage des données de la manière souhaitée, afin d'obtenir la data visualisation la plus **lisible** possible.

Dans notre cas, nous avons intégré kepler.gl dans notre propre page html, afin de directement charger et afficher notre jeu de données de la manière que nous voulions via le code. De cette manière l'utilisateur n'a aucune action à effectuer, il peut directement consulter les données affichées sur la carte. D'aventure, si celui-ci souhaite changer l'affichage des données, il peut tout de même avoir accès aux différents paramètres d'affichage en cliquant sur la petite flèche en haut à gauche de la carte, et ainsi changer les différents paramètres via l'interface graphique.

## Calculs des coordonnées : Python Getting coordinate grid per country et Wicket

Pour obtenir les coordonnées des différents points, il a fallu employer une méthode précise qui va être décrite ci-dessus.

La première étape fut de choisir un territoire dont nous voulions récupérer les points, ensuite nous avons utilisé **Wicket**. C'est une bibliothèque Javascript légère qui lit et écrit des chaînes de caractères WKT (Well-Known Text). Elle peut également être étendue pour analyser et créer des objets géométriques à partir de divers cadres cartographiques, tels que Leaflet, l'API JavaScript d'ESRI ArcGIS et l'**API de Google Maps**. Grâce à elle nous avons pu tracer un carré autour du territoire puis récupérer les coordonnées des points en haut à gauche et en bas à droite.

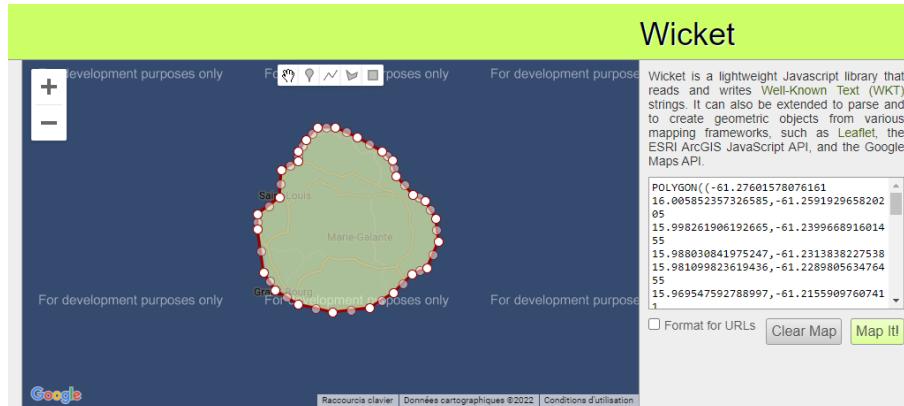


Ensuite nous avons utilisé du **Python** ainsi que la bibliothèque **Shapely** pour la suite de la méthode. Shapely nous a permis d'avoir accès à des outils permettant de créer la grille ainsi que les points qui la composent. Les points du carré servent dans une fonction trouver sur le sujet [Getting coordinate grid per country](#) du forum Geographic Information Systemes, permettant d'obtenir une grille de points faisant la surface du carré tracé sur le territoire. Cette fonction permet aussi de régler l'espacement entre les points, plus celui-ci est petit, plus il y a de points dans la grille et plus la suite du procédé peut être précise.



*Exemple de grille créée grâce à la fonction*

Ensuite, il faut retourner sur Wicket afin de tracer les frontières du territoire grâce à l'outil Polygone. Une fois cette étape faite, les coordonnées peuvent être récupérées et placées dans le fichier Python. On peut alors créer un Polygone avec eux grâce à la fonction **wkt.load** de Shapely permettant de le créer à partir de chaînes de caractères WKT.



Après cette étape un code Python est utilisé, celui-ci permet de savoir si les points de la grille carrée se trouvent dans la zone définie par les coordonnées des frontières. Si c'est le cas, les points sont ajoutés à un tableau.

L'étape suivante n'est pas indispensable, mais nous avons vérifié sur Wicket, grâce à l'option Map It, si les points étaient bien placés et correspondaient à ce que nous voulions.



Pour finir, nous avons créé un fichier Python contenant une fonction pour créer un fichier **csv**. La fonction prend en paramètre une grille de point et un nom de fichier, elle sépare ensuite les longitudes et latitude de tous les points présents dans la grille et les mets dans une liste. Cette liste est ensuite utilisée pour écrire le csv avec le nom passé en paramètre.

The terminal window shows the creation of a CSV file named 'reunion.csv'. The file contains 20 entries, each consisting of a line number followed by a pair of coordinates (lon, lat). The coordinates are identical for all entries.

```

chage  Atteindre  Exécuter  Terminal  Aide  reunion.csv -
guadeloupe.py  reunion.csv  mayotte.py
IMAC > csv > reunion.csv
1 lon,lat
2 55.55282193720964, -21.363983021238944
3 55.60282193720964, -21.363983021238944
4 55.65282193720964, -21.363983021238944
5 55.702821937209634, -21.363983021238944
6 55.75282193720963, -21.363983021238944
7 55.45282193720965, -21.313983021238943
8 55.502821937209646, -21.313983021238943
9 55.55282193720964, -21.313983021238943
10 55.60282193720964, -21.313983021238943
11 55.65282193720964, -21.313983021238943
12 55.702821937209634, -21.313983021238943
13 55.75282193720963, -21.313983021238943
14 55.80282193720963, -21.313983021238943
15 55.352821937209654, -21.263983021238943
16 55.40282193720965, -21.263983021238943
17 55.45282193720965, -21.263983021238943
18 55.502821937209646, -21.263983021238943
19 55.55282193720964, -21.263983021238943
20 55.60282193720964, -21.263983021238943

```

Cette méthode a été utilisée séparément pour les territoires de la France, Corse, Réunion, Mayotte, Martinique, Guadeloupe, Guyane et Marie-Galante.

# Calcul mathématiques des distances sur une sphère : Haversine

Pour notre projet nous avons décidé de mettre en avant les **disparités** entre les différents points du territoire en mettant en avant la distance à la plus proche structure de la DGFIP pour chaque point du pays. Par la suite ces distances serviront à donner la hauteur du bâton placé en ce point. La première étape pour ce faire est de calculer la distance entre deux points sur le globe. Pour calculer les distances entre les points du territoire et les différentes structures nous avons utilisé la **méthode de Haversine**. Cette méthode permet de mesurer la distance (à vol d'oiseau) entre deux points sur une sphère à partir de leurs **longitudes** et **latitudes**.

La formule donne :

En prenant deux points sur une sphère, le haversine de l'angle au centre est donné par

$$\text{hav}\left(\frac{d}{r}\right) = \text{hav}(\varphi_2 - \varphi_1) + \cos(\varphi_1) \cos(\varphi_2) \text{hav}(\lambda_2 - \lambda_1)$$

où on a :

- hav est la fonction **haversine** :
- *d* est la distance du grand cercle entre les deux points
- *r* est le rayon de la sphère,
- $\varphi_1, \varphi_2$ : latitude du point 1 et latitude du point 2, en radians
- $\lambda_1, \lambda_2$ : longitude du point 1 et longitude du point 2, en radians

Puis en implémentant en python on a (code extrait du fichier calculs.py)

```
#calcul de la distance entre deux points GPS
def haversine(coord1, coord2):
    R = 6372800 # Rayon de la Terre (en mètre)
    lat1, lon1 = coord1
    lat2, lon2 = coord2

    phi1, phi2 = math.radians(lat1), math.radians(lat2)
    dphi      = math.radians(lat2 - lat1)
    dlambda   = math.radians(lon2 - lon1)

    a = math.sin(dphi/2)**2 + \
        math.cos(phi1)*math.cos(phi2)*math.sin(dlambda/2)**2

    return 2*R*math.atan2(math.sqrt(a), math.sqrt(1 - a))
```

Par la suite nous trouvons la **distance minimum** en calculant la distance entre le point considéré et toutes les structures de la DGFIP.

De cette manière, le projet peut être remis à jour de manière assez simple. En effet, si une nouvelle structure est ajoutée il suffit de recalculer sa distance avec les points du territoires, pour les points pour lesquels cette distance est inférieure à la précédente on la remplace sinon rien ne changera.

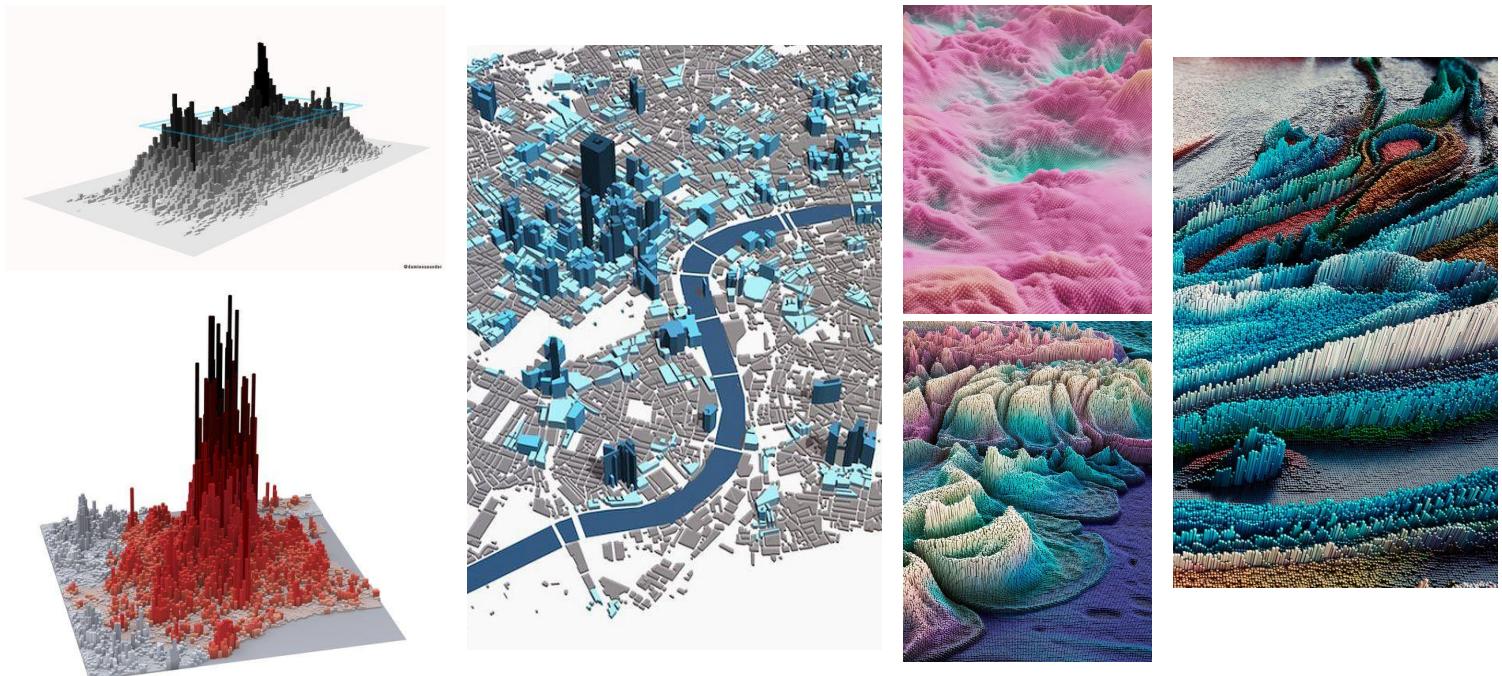
## MOODBOARD

Plusieurs visuels très différents nous ont inspirés pour l'esthétique de notre projet.

Tout d'abord l'idée de **monochrome** de couleur et de variation de la luminosité pour indiquer un pic plus ou moins grand.

Puis pour les **couleurs** nous avons été inspirés par ces images de villes en "thème sombre négatif" dans les bleus et gris.

Pour contraster avec la rigidité des données, des figures plus **organiques** comme ces peintures nous ont aussi inspirées.

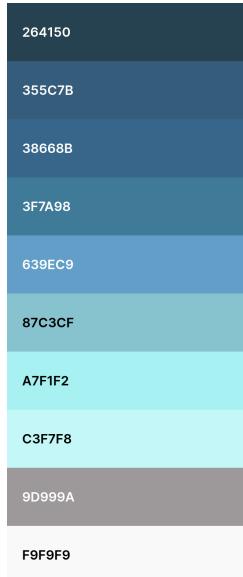


# CHARTE GRAPHIQUE

## Typographie : Montserrat

Pour la typographie, nous avons choisi la Montserrat car elle est simple et épurée. Le fonctionnel est mis en avant grâce à sa grande visibilité. Comme il n'y aura pas beaucoup d'informations sur notre carte, nous souhaitons que ces dernières soient lisibles tout en restant discrètes et ainsi ne détournent pas l'attention de ce qui est important.

## Palette de couleur :



Nous avons choisi d'utiliser la couleur **bleu** car c'est un symbole de vérité, comme l'eau limpide qui ne peut rien cacher. Cela permet de mettre en avant une **vérité brute** sans détour. Le bleu est aussi une couleur **universelle** parlant à toutes les générations. C'est aussi une des couleurs de la France (nous avons préféré le bleu au rouge car cela aurait pu avoir une interprétation trop violente et aurait pu biaiser la lecture des données).

Nous utilisons un **monochrome / camaïeu** afin de montrer l'unité ou le vœux d'**unité** qui existe en France.

*Visuel sur Kepler.GL et esthétique possible (exemple avec un autre jeu de donnée)*



Kepler nous permet de jouer avec la hauteur et donc de proposer une notion de distance afin de montrer les différentes inégalités quant à la répartition des structures sur le territoire.

# DESCRIPTION DE NOTRE PROJET

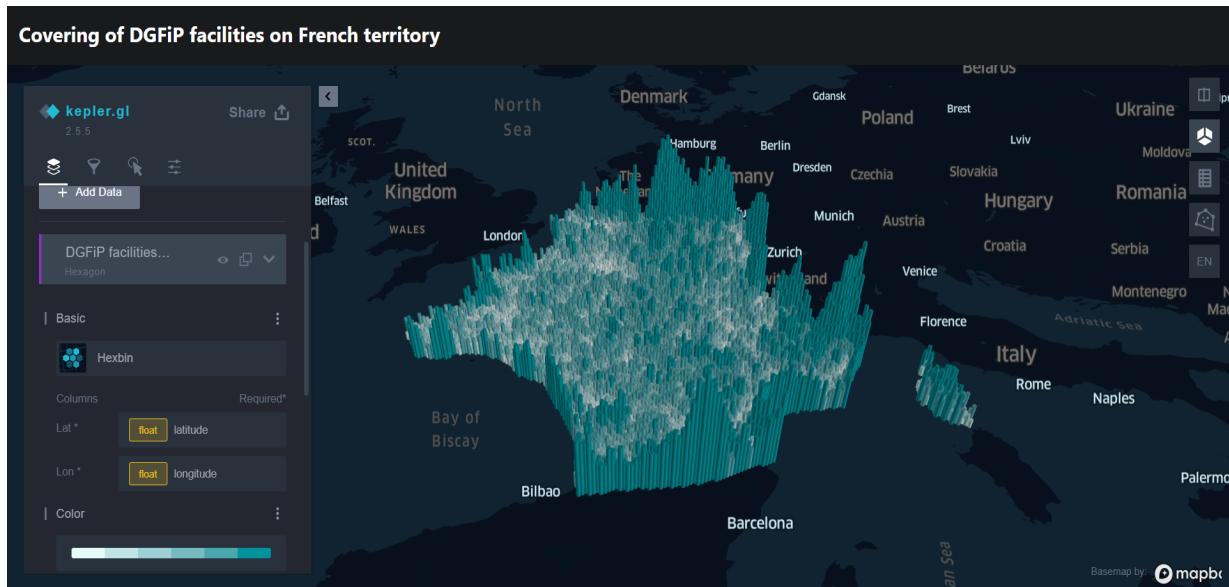
Pour ce projet nous avons voulu faciliter la visualisation des **disparités** à l'accès au services.

Pour cela nous avons choisi de mettre en avant les **différences** dans les distances de trajets selon le domicile de chacun. Pour cela nous avons, pour chaque point de la carte, effectué le calcul de la distance au point le plus proche pour chaque point de la France. Visuellement cette distance est représentée par un **cylindre** vertical de la hauteur correspondante à la distance à parcourir.

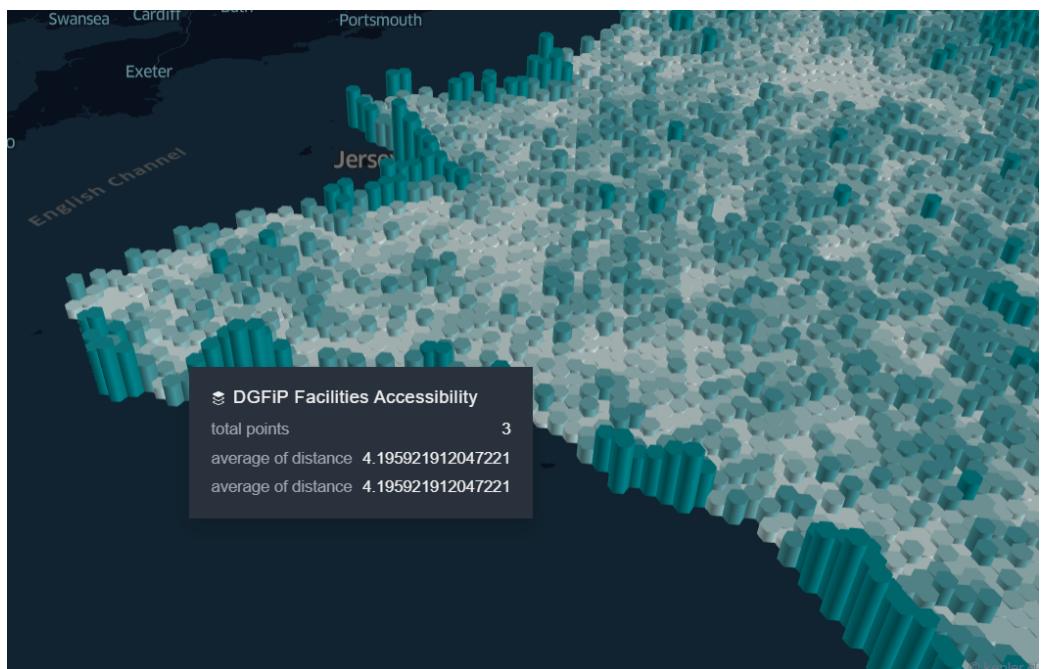
Les points les moins bien desservis sont directement mis en avant par des **pics** bien plus hauts que les autres. La **carte en 3D** permet une exploration plus facile du territoire et une **lecture en 2 temps** (en nuance de couleur lorsque la carte est vu du dessus puis en contraste de relief lorsqu'on passe en 3D)

## RENDU

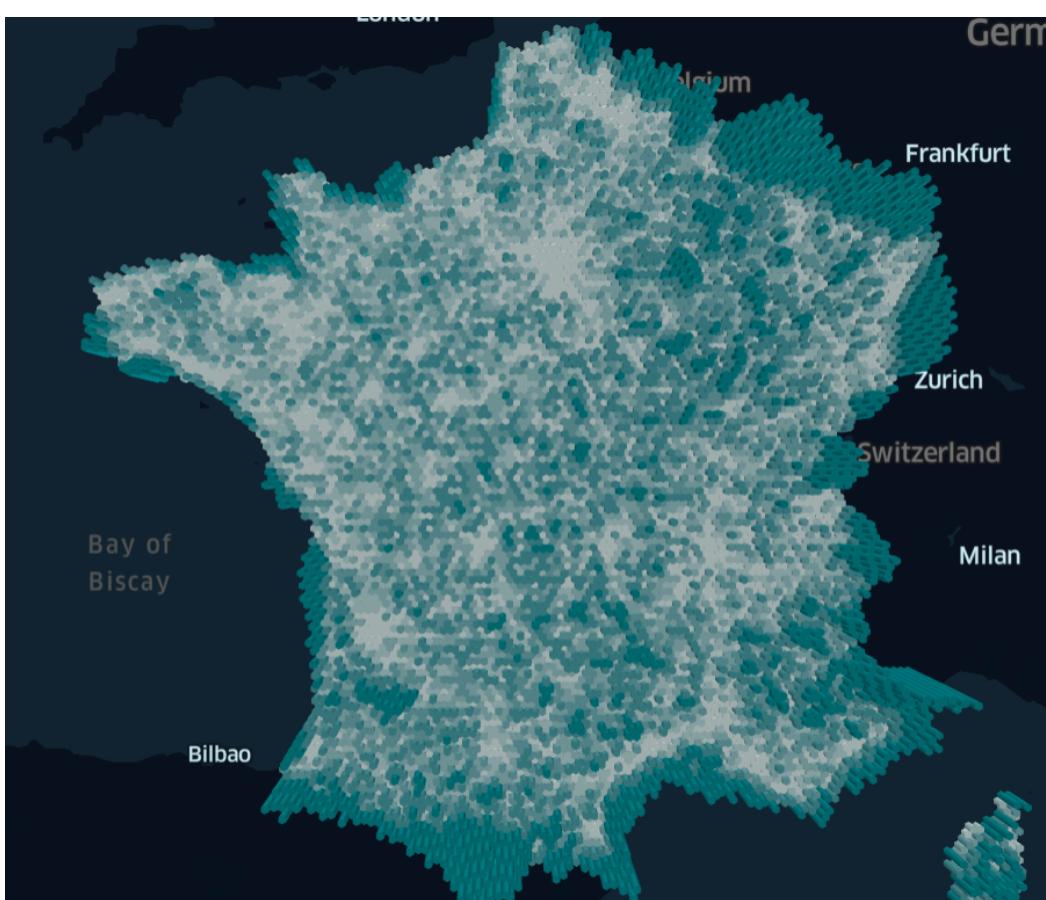
Notre visualisation finale est donc hébergée sur un site Web. Les options de visualisations proposées par Kepler sont toujours accessibles. Nous avons néanmoins pensé nos données pour être affichées avec le mode **Grid** ou **Hexbin**



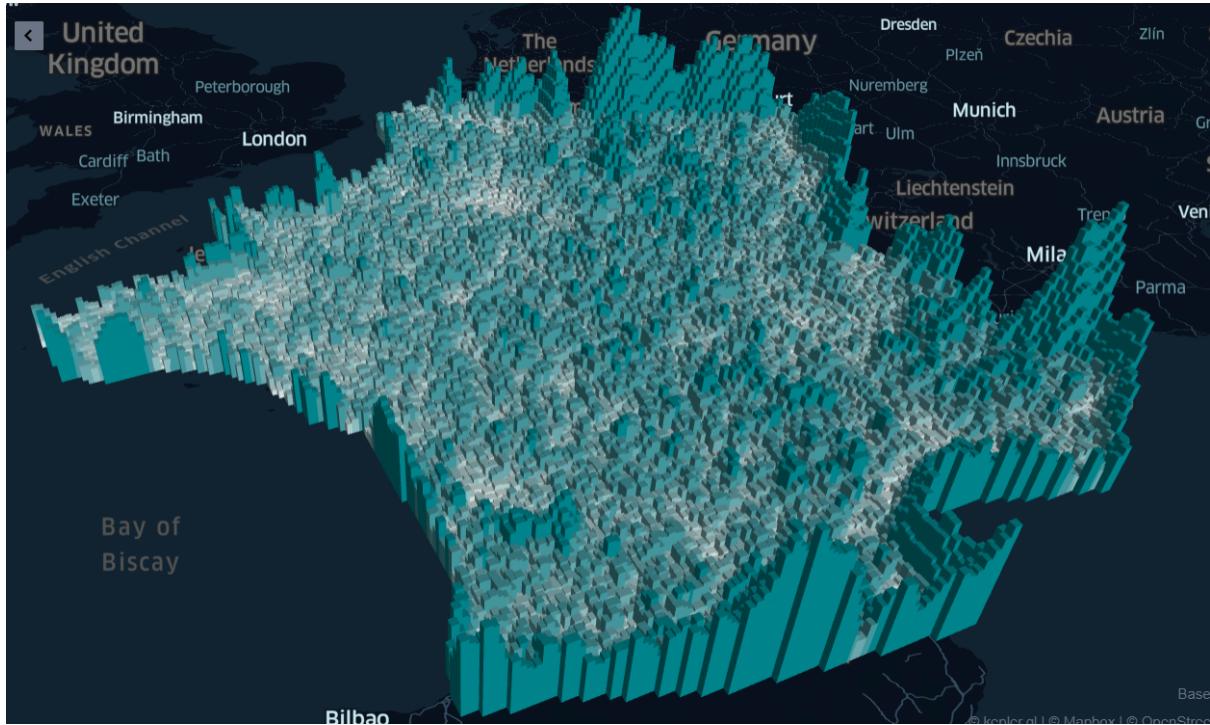
Visuel de notre projet : Kepler et ses options de visualisation à gauche



*Détails des données au survol du point*



*Carte dans sa visualisation 2D. les pics claires correspondent aux temps les plus courts*



*Visualisation avec le mode grid (les pics sont de bases carrées)*

Lien vers notre projet en ligne : <https://tanguylorent.com/INTERACTIVITY/>

Lien du git : <https://github.com/hackaton-dgfip-2022/IMAC>

## POUR ALLER PLUS LOIN

Lorsque nous avons pensé ce projet nous avons réfléchi à ce qui pouvait être ajouté à ce projet avec plus de temps et nous avons relevé plusieurs points.

### **Les filtres:**

Kepler.GL est un module très riche et permet de filtrer les données entrées ainsi nous pourrions envisager d'afficher les distances selon l'heure et/ou le jour.

Malheureusement il manquait les horaires d'ouvertures sur certaines structures et nous avons donc décidé de nous focaliser sur d'autres points. De plus cela rajoute une grosse quantité de calculs car il faut refaire les calculs de distances pour chaque tranche horaire et nous ne possédons pas les ressources calculatoires nécessaires sur nos machines.

Un autre filtre qui pourrait être ajouté est le type de démarche recherchée.

### **Affichage par département/région:**

En utilisant la méthode décrite précédemment pour trouver les points d'un territoire on pourrait aussi récupérer les coordonnées des points par département et/ou région et ainsi permettre un affichage montrant uniquement le lieu demandé. Cela demanderait encore une fois de refaire des calculs de distance car on ne devrait prendre en compte que les structures qui se trouvent dans la région/département en question.

### **Amélioration des contours**

Sur le rendu final on peut voir que l'on considère certains points hors de la france qui affichent du coup des pics très élevés. Ce problème pourrait être réglé par des calculs plus précis des contours, ce qui demande encore des ressources calculatoires que nous n'avons pas.

### **Affichage des temps de trajet**

La dernière amélioration à laquelle on à pensé est liée à la valeur utilisée pour choisir la hauteur des bâtons. Pour l'instant on utilise la distance à vol d'oiseau jusqu'à la structure la plus proche mais on pourrait imaginer lier notre site à l'API de google map et récupérer des informations sur les temps de trajets entre les deux points selon le transport utilisé. Cela permettrait d'aller plus loin dans la mise en avant des disparités, notamment entre les zones rurales et urbaines, surtout si on veut se déplacer en transport en commun.