



Refonte et amélioration d'une application de cartographie SIG

Rapport de stage

ROBIN RULLO

DIPLOÔME UNIVERSITAIRE
D.U. 4.0.2

PROMOTION 2020 – 2021 – UHA 4.0.2

14/02/2022 – 12/08/2022



Tuteurs académique :

MOUNIR ELBAZ
mounir.elbaz@uha.fr

Tuteur entreprise :

EL MAHDI SAHI
m-sahi@logitud.fr

Remerciements

Je souhaite tout d'abord remercier Guillaume LOOS, responsable des développements, qui m'a permis d'intégré dans l'équipe de Recherche et Développements.

Je tiens à remercier El Mahdi SAHI, responsable du service R&D ainsi que mon collègue Mohamed TAMA, ingénieur géomaticien et développeur, toujours disponible et qui m'a pas mal forcé la main pour rédiger ce rapport en temps et en heure. Merci à eux de m'avoir suivi et fait confiance tout au long du stage.

Je remercie également tous les développeurs, les hotliners, les formateurs, et plus généralement tous ceux qui ont pris le temps de répondre a mes questions et avec qui j'ai pu échanger et ainsi progresser.

Table des matières

1	Introduction	1
2	Organisme d'accueil	2
3	Le stage	3
3.1	Présentation du contexte	3
3.2	La géomatique et le SIG	3
3.3	Étude de l'existant	4
3.3.1	Le lien entre les géométries et les objets dans les applications	5
3.3.2	État de l'art	5
3.4	Définition du besoin	6
3.5	Développement du projet et difficultés rencontrées	8
3.5.1	Prototypage	8
3.5.2	Choix des technologies et structure	9
3.5.3	Implémentation de la maquette	10
3.6	Déploiement	16
3.7	Améliorations et perspectives	17
	Glossaire	18
	Conclusion	19
	Annexe	

Table des figures

1	Les 1032 sites constituant le réseau de base français pour maintenir le RGF93	3
2	Projection Lambert conique conforme à gauche, Mercator à droite © Tobias Jung	4
3	Architecture SIG	6
4	Choix de la vue selon le type de géométrie	7
5	Place monopolisée par les menus	7
6	Mapillary	9
7	Proposition retenue	9
8	App Shell	10
9	Barre latérale	11
10	Capture d'écran d'une carte d'un type et en-dessous celle d'un secteur	11
11	Composant de recherche de types à gauche et composant de visualisation du contenu du type à droite	11
12	Architecture basée sur les flux	12
13	Map Manager	13
14	Icon « Clock » de Clarity non traité vers l'icone traité	14
15	Vue de l'import d'objets	15

1 Introduction

Logitud Solutions est une entreprise spécialisée dans l'édition de logiciels pour les collectivités, dans les domaines de la population et de la sécurité, depuis 30 ans. Le web s'étant beaucoup développé ces dernières années, les logiciels sont devenus difficiles et coûteux à maintenir. L'entreprise a alors amorcé depuis quelques années la réécriture de ses applications lourdes liées au secteur de la sécurité (polices municipales) qui nécessitent une installation sur un poste de travail, vers des applications web fonctionnant sur un navigateur web ainsi que des applications pour mobiles.

L'ensemble des applications clients lourds puis applications web font appel à des données géo-référencées. Le rôle de l'application web **SIG MAP-MANAGER** qui permet d'administrer les données géographiques. L'entreprise souhaite faire évoluer l'application ainsi que la rendre plus ergonomique. Pour répondre à de nouveaux besoins, une refonte de l'application est nécessaire.

Attiré et spécialisé dans les systèmes d'information géographique depuis le début de la formation à l'UHA 4.0, je souhaitais, en déposant ma candidature, assurer mes connaissances en géomatique dans un milieu professionnel, entouré d'experts pouvant me guider et échanger leurs connaissances.

Le rapport couvre la présentation de l'organisme dans lequel le stage a eu lieu, puis une présentation du stage. Enfin, une conclusion boucle le rapport.

2 Organisme d'accueil

Logitud Solutions est une société par actions simplifiée dont le siège social est situé dans la ZAC du Parc des Colline de Mulhouse – Didenheim. Elle compte également deux autres agences, l'agence centre à Saint-Avertin (37), et l'agence sud à Saint-Rémy-de-Provence (13). Elle est constitué de 90 employés dont 30 développeurs.

Son secteur d'activité est l'édition d'outils numériques déstinés aux collectivités locales (communes, communautées de communes, villes). Elle distribue aujourd'hui trois gammes de logiciels, chronologiquement :

La gamme population – Elle est tournée vers la gestion administrative des collectivités. Elle facilite le travail des agents d'état civil et leurs échanges avec les administrés avec des logiciels tels que Siècle, SuffrageWeb, Éternité, Avenir ou encore Populis.

La gamme sécurité Elle est orienté vers la gestion des métiers de la police (organisation, gestion des fourrières, géo-verbalisation) et prévention de la délinquance (geo-prévention délinquance et des incivilités).

La gamme e-administration Elle regroupe des services en ligne et mobiles à l'attention des citoyens.

L'entreprise est un acteur majeur du marché dans cette gamme de logiciels. Elle équipe un tier des villes de plus de 5 000 habitants en produits d'état-civil, et les quatre cinquième en produits de sécurité (polices municipales).

La méthodologie de travail adopté dans les équipes de développement métier suit les principes du **framework SCRUM** couplé à la méthodologie **Agile**. Cependant l'équipe de R&D dont est rattaché le pôle de développement cartographique bénéficie d'énormément de liberté dans son management et où aucune stratégie de management spécifique n'est appliquée.

3 Le stage

3.1 Présentation du contexte

Afin de s'adapter aux services proposés par la concurrence ainsi qu'aux technologies actuelles, Logitud est repartie de zéro en réécrivant les applications qui était alors jusque là des clients lourds en applications web (clients légers). De nombreuses applications des suites métiers (gamme population, sécurité, etc...) interagissent avec des données géo-référencées. MAP MANAGER est l'application **SIG** permettant d'administrer ces différentes données. Elle est maintenue par l'équipe en charge de l'infrastructure géographique.

3.2 La géomatique et le **SIG**

La géomatique ou « la géographie appliquée à l'informatique », est la discipline regroupant les pratiques qui permettent de collecter, analyser et diffuser des données géographiques par l'informatique. L'application MAP MANAGER s'inscrit principalement dans la diffusion des données mais aussi dans la collecte avec la possibilité de création et modification d'objets géographiques.

Afin de se repérer et de localiser l'information sur la surface terrestre, il est nécessaire d'utiliser un système de positionnement comprenant :

la définition d'un référentiel dont son but est de fournir aux utilisateurs des points stables et matérialisés par des bornes de coordonnées connues. En France, la norme est le référentiel RGF (Réseau Géodésique Français) 93. Cependant, la norme internationale est le WGS (World Geodetic System) 84, utilisé par les américains et associé au système GPS.



FIGURE 1 – Les 1032 sites constituant le réseau de base français pour maintenir le RGF93

le choix d'un système de projections et de coordonnées dont le but est de projeter l'image de la terre assimilé à un elipsoïde en une surface plane. Encore une fois, en France, nous utilisons la projection Lambert 93. Cependant, la plupart des cartes numériques mises à disposition du grand public utilisent la projection WGS84 Web Mercator.

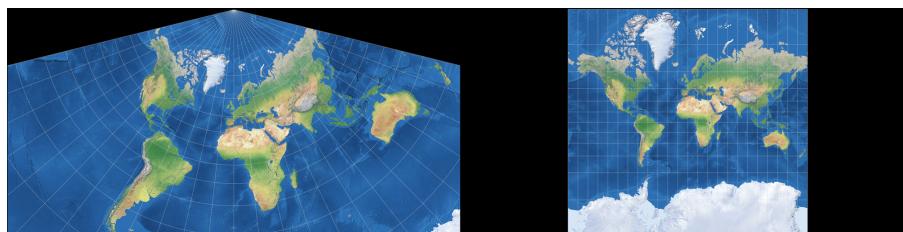


FIGURE 2 – Projection Lambert conique conforme à gauche, Mercator à droite
© Tobias Jung

MAP-MANAGER est une application **SIG**. Le **SIG**, pour Système d'Information Géographique, est un système d'information qui intègre, stocke, analyse et affiche l'information géographique qui est de la donnée localisée sur le territoire. Cette donnée peut être :

Géométrique : La donnée décrit la forme et la position (points, lignes, polygones), repéré dans un système de projection retenu et donc superposable avec d'autres données.

Attributaire : La donnée attributaire fournit des informations complémentaires permettant de caractériser la donnée géométrique, de type numérique, texte, date, etc... .

Semiotique : La donnée sémiologique fournit les informations pour représenter les données géométriques sur la carte (taille, couleur, pictogrammes, etc...).

3.3 Étude de l'existant

L'application MAP MANAGER existe déjà mais qui ne correspond plus aux besoins en terme d'ergonomie et de fonctionnalités. C'est une Single Page Web Application (SPA – application web à page unique). A mon arrivée, elle avait déjà subit trois refontes car plusieurs besoins qui n'avaient pas été exprimés au début du développement se sont ajoutés au fur et à mesure de l'utilisation de l'application.

L'application a également été homonyme en librairie Angular, appelée MAP-VIEWER, dont son but n'est plus d'administrer mais simplement d'afficher les données cartographiques dans les **applications métier**. Cette librairie, est incluse dans la librairie de composants graphiques partagés utilisé par l'entreprise, WEBUI-CORE,

afin d'uniformiser et simplifier l'affichage des données dans les autres applications. Cette librairie est une librairie Angular avec un composant qui permet d'avoir la même carte et interactions que celles développées dans map-manager. Il a également fallut faire une réécriture car les interfaces n'étaient pas unifiées de base mais devait prendre en compte les mêmes besoins que MAP MANAGER mais cette fois en gardant la même base car il ne fallait pas produire de Breaking Changes (Modification cassantes nécessitant une adaptation du coté des application l'ayant implémenté).

J'ai réalisé la première semaine un document rendant (cf. [annexe 2](#)) compte de l'état des fonctionnalités développées en suivant l'approche du « Manual Testing », pratique qui consiste à tester toutes les fonctionnalités manuellement sur l'interface web afin de tester entièrement les fonctionnalités. Ce document m'a permis de définir le point de départ et de comprendre le besoin des clients. Il m'a permis de mettre en évidence les points qui suivent.

3.3.1 Le lien entre les géométries et les objets dans les applications

Il y a trois catégories d'objets géo-référencés qui sont constitués des trois différentes géométries présentées précédemment :

- Les **secteurs** représentés par une géométrie polygonale
- Les **POIs** (Point of interest – Points d'intérêts) représentés par le point
- Les **itinéraires** représentés par la géométrie linéaire.

Dans le domaine métier géographique, les collègues ont fait le choix d'organiser ces géométries dans un ensemble de types. Un type est défini par un nom (ex : « Stationnement payant »), une couleur, ainsi qu'une icône. La couleur peut être surchargée dans l'objet géométrique que contiendra le type tandis que l'objet possèdera forcément l'icône du type. Ces types peuvent être rattaché à un ou plusieurs contextes métiers propre à un module d'une application de la suite logicielle. Les contextes métiers sont gérés par le service LABELS qui est commun à toutes les applications, qui permet principalement de personnaliser les données de l'interface utilisateur à la guise du client mais également de contenir certains paramètres métiers liés aux applications.

3.3.2 État de l'art

Toute l'infrastructure SIG de l'entreprise est déjà en place et fonctionnelle. Elle est composée de plusieurs mini-services divisé en deux thématiques, cf. figure [3](#). La première partie entourée en orange est l'administration des objets géographiques. C'est également celle sur laquelle j'ai travaillé. La deuxième est la recherche d'adresses.

La première partie est composée de quatre modules

- GEOTOOBOX, le serveur de traitement et stockage cartographique est le module principal.

- GEOSERVER est un serveur de données cartographiques open-sources. Il est consommé par le serveur GeoToolbox pour réaliser tout les traitements géospatiaux comme le [reverse-geocoding](#).

La seconde partie est composée de trois modules : Le serveur ADDRESSES [proxy](#) les services de recherche d'adresses. Pour la recherche d'adresses en France, nous utilisons un serveur de recherche d'adresse ADDOK avec les données de la Base Nationale d'Adresses fournie par l'État. Pour les recherches d'adresses en dehors de la France, nous consommons un service open-source basé sur les données d'OpenStreetMap.

MAP-MANAGER est l'application web permettant d'administrer les données du serveur GeoToolbox. On peut également y rechercher des adresses même si cette fonctionnalité est plutôt utilisé dans les [applications métier](#).

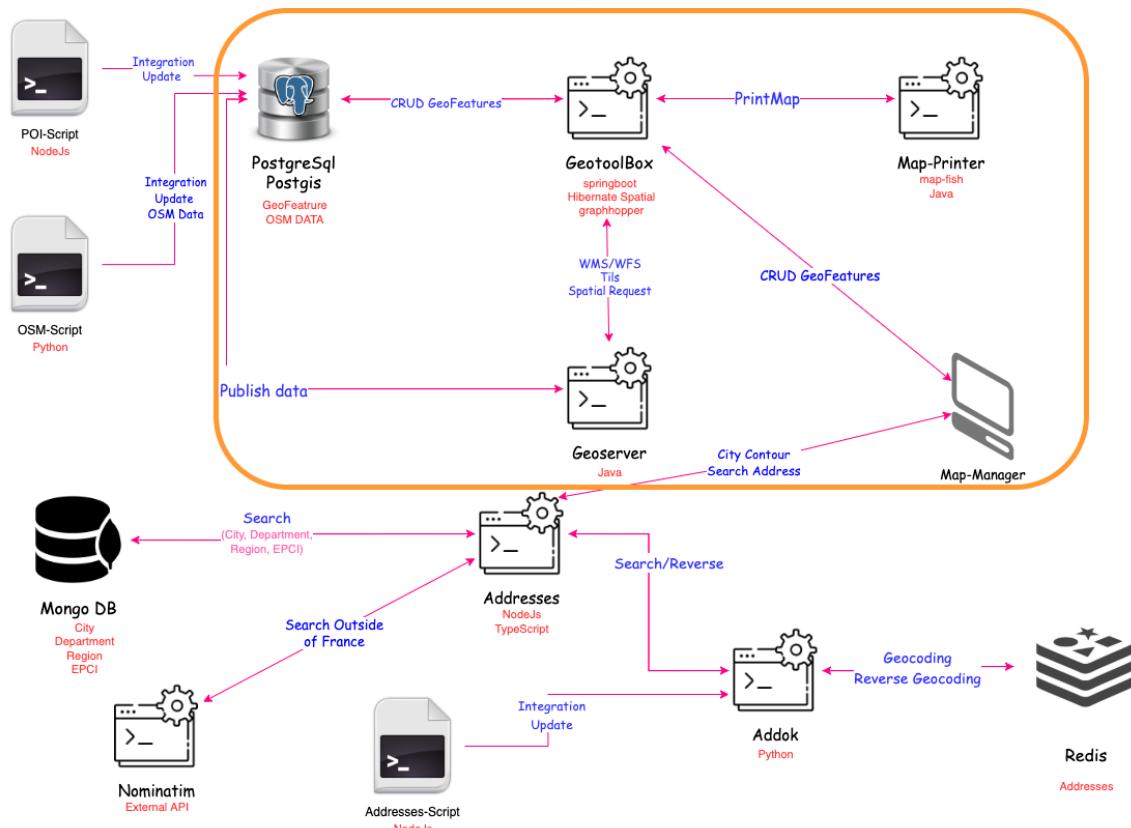


FIGURE 3 – Architecture SIG

3.4 Définition du besoin

MAP-MANAGER répond au besoin d'administration de secteurs, itinéraires et poi. Cette fonctionnalité a été séparé en trois modules, sur trois vues différentes, ce

qui n'est plus le cas aujourd'hui. Il ne doit plus y avoir de séparations en fonction du type de géométrie.



FIGURE 4 – Choix de la vue selon le type de géométrie

Une fois un type de géométrie sélectionné, il est possible de sélectionner un ou plusieurs types ou contextes. Des géométries s'affichent et un tableau souvre sur la partie médiane basse de l'écran, listant les géométries correspondants aux critères de recherche. Sur de petits écrans, la place est monopolisée par le menu de recherche et ce tableau :

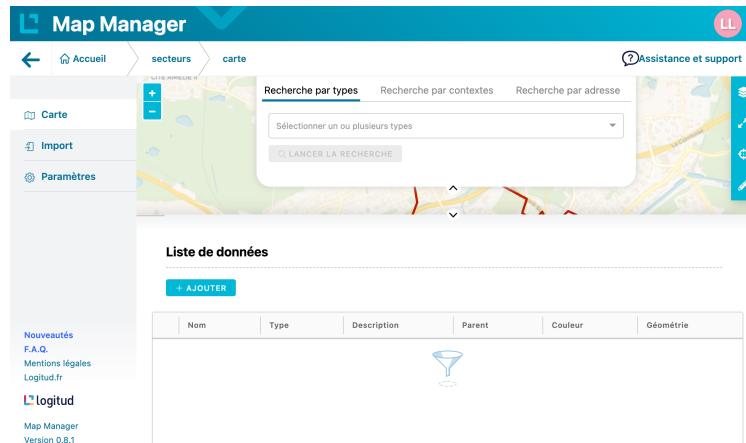


FIGURE 5 – Place monopolisée par les menus

En ce qui concerne ensuite la modification des geométries, on peut soit en ajouter, soit les modifier avec des outils de la carte très basiques. On peut également consulter les données qui y sont référencées.

Une page paramètre dans l'application permet de réaliser un SCRUD (Search, Create, Read, Update, Delete) sur les types métiers. Une autre section de l'application permet également d'importer une collection de données géographiques dans un type métier déjà existant (à nouveau avec la contrainte de séparation des types de géométrie). J'ai également pu remonter un certain nombre de comportements indésirés ou bugs.

Pour interagir avec les objets géographiques, il faut passer par le serveur GeoToolbox. Il permet de créer les types, les objets géographiques et de les modifier par la suite. Le serveur GeoToolbox est développé en parallèle et indépendamment de Map-Manager par un collègue expert géomaticien.

Avant de commencer la réécriture de l'application, nous avons déterminé les principaux besoins sur lesquels se focaliser pour cette nouvelle version :

1. L'application doit être dans un premier temps iso-fonctionnelle.
2. Afficher toutes les catégories d'objets géographique sur une seule carte.
3. Toutes les fonctionnalités dans une seule vue carto-centrée.
4. L'application doit être ergonomique pour l'utilisateur qui fréquente peu l'application et qui n'est pas à l'aise avec les **SIG**.
5. L'application devra utiliser les librairies communes aux autres applications.

Pour permettre à l'application d'unifier les secteurs, les POI et les itinéraires en une seule entité, un collègue s'est attelé en parallèle à la refactorisation du serveur GeoToolbox.

3.5 Développement du projet et difficultés rencontrées

Nous verrons d'abord le prototypage de la nouvelle version. Nous verrons ensuite la création et l'implémentation des composants de l'interface utilisateur. Nous verrons l'interaction des données avec les autres services de la suite logicielle. Nous analyserons ensuite l'implémentation de la carte, comprenant la génération du style (sémiologie de la carte) puis les interactions avec celle-ci. Nous verrons par la suite la fonctionnalité d'import d'objets géographiques pour enfin finir par le déploiement en production et les améliorations qu'il reste à prévoir.

3.5.1 Prototypage

J'ai débuté la refonte par la réalisation d'une maquette pour l'interface utilisateur sur un logiciel de prototypage, **Adobe XD**. Mon collègue m'a suggéré de m'inspirer du site [Mapillary](#), cf. figure 6.

Cependant, le design de la maquette ne correspondait pas à celui des autres applications de Logitud. J'ai donc continué le prototypage en me basant sur le système de design Clarity UI de VMWare afin d'uniformiser l'interface avec celles des différentes applications car la librairie de composant partagée de l'entreprise, WEBUI-CORE, utilisée par les autres applications.

Nous avons ensuite organisé une réunion avec le référent UI/UX, le responsable du service R&D, le responsable des développements et également un responsable de projet de la hotline Logitude afin de valider l'implémentation du besoin métier ainsi que l'ergonomie à l'utilisation. Nous avons retenu la proposition suivante (cf. figure 7) :

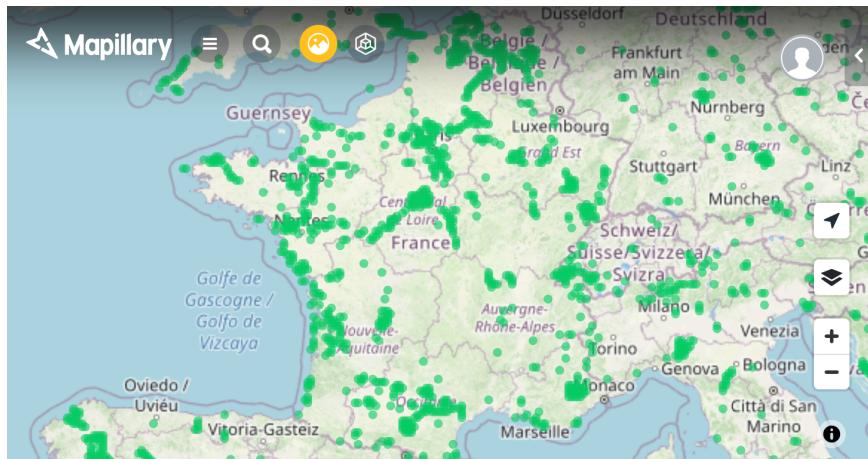


FIGURE 6 – Mapillary

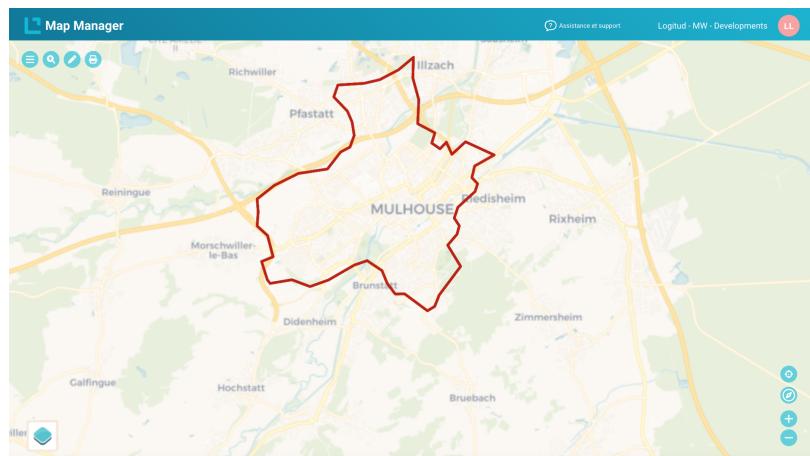


FIGURE 7 – Proposition retenue

3.5.2 Choix des technologies et structure

Le projet est basé sur le framework Web Angular, dans sa version 9. Nous avons été contraint à ce choix car toutes les applications de l'entreprise sont développées avec ce framework sur cette version et nous ne pouvons pas monter la version car plusieurs librairies communes, notamment WEBUI-CORE, sont bloquées en version 9 à cause de trop nombreux breaking changes à corriger lors de la montée vers la version 10.

En ce qui concerne le Web-mapping, nous avons pris la décision de continuer d'utiliser la librairie OpenLayers permettant d'afficher la carte dynamique. Nous en avons une bonne connaissance, elle est open-source, très mature ainsi que suffisante pour répondre à nos besoins actuels.

Nous avons décidé pour la réécriture, d'initialiser un nouveau projet Angular et de tout réimplémenter en suivant l'architecture que nous avions défini afin de rendre

l'application maintenable :

```
src/app/  
+-- config  
+-- core  
|   +-- http  
|   +-- layout  
+-- enums  
+-- interfaces  
+-- modules  
|   +-- geo-entity  
+-- services  
|   +-- external  
+-- shared  
|   +-- directives  
|   +-- map  
|   +-- modal  
+-- utils
```

3.5.3 Implémentation de la maquette

Map-Manager intéragis avec de nombreux services de la suite logicielle que nous découvrirons plus tard. Nous avons décidé de se focaliser de prime abord sur l'intégration de la maquette dans l'application Angular. C'est pourquoi toutes les models de données fournis par les services externes ont été mockés : une réponse « type » avec des données statiques est renvoyé au lieu de consommer réellement le service externe.

Création des composants de la maquette

L'application étant la seule carto-centré dans la suite d'applications, aucun composant n'est intégré à la librairie WEBUI-CORE excepté l'[App Shell](#).



FIGURE 8 – App Shell

J'ai donc débuté par l'ajout des composants de la barre latérale avec des données mockés :

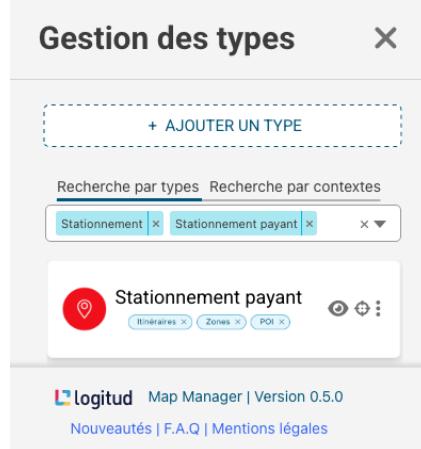


FIGURE 9 – Barre latérale

Tout d'abord par la création des « cards » permettant de lister les types et les objets géographiques contenus dans les types :



FIGURE 10 – Capture d'écran d'une carte d'un type et en-dessous celle d'un secteur

J'ai poursuivi par l'implémentation du composant d'affichage des objets géographiques en utilisant les cards créés précédemment ainsi que le composant de recherche de types avec leurs recherches :



FIGURE 11 – Composant de recherche de types à gauche et composant de visualisation du contenu du type à droite

J'ai rencontré quelques difficultés lors du changement de vue dans la sidebar vers le composant d'affichage des objets géographiques. En effet, il provoque la destruction du composant d'affichage des types. Tous les filtres de recherches étaient donc détruit et réinitialisé lorsqu'on revenait de la vue du contenu d'un type vers la

vue des types qui contenait l'état du filtre. Je me suis retrouvé dans la problématique qu'était facebook il y a quelques années et dont découle l'architecture basée sur les flux.

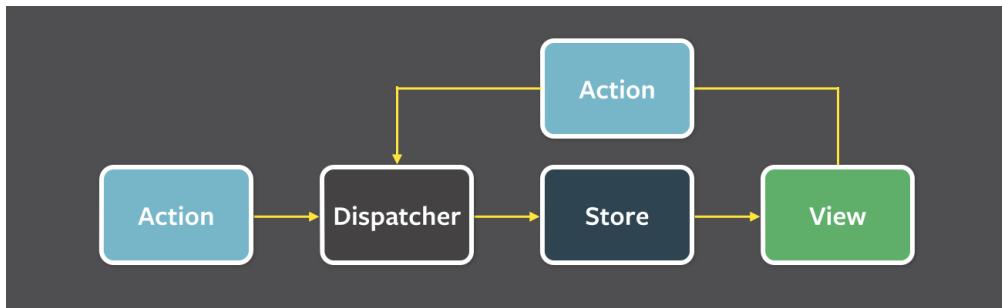


FIGURE 12 – Architecture basée sur les flux

Le principe de flux simplifie la dépendance des composants entre eux. Dans le cas ci-présent, l'état des filtres est gardé dans un store global dans l'application qui est détruit avec l'application. Le composant est mis à jour et re-rendu lorsque cet état est modifié avec le pattern *Publish-Subscribe*.

Pour résoudre ma problématique, j'ai proposé de mettre en place cette architecture dans l'application. Cependant j'étais le seul à l'aise avec une architecture basé sur les flux dans l'entreprise et d'autres solutions existent déjà dans Angular. Les deux autres solutions à mon problème sont de garder l'état du filtre dans le composant parent ou dans un service. Ces deux solutions sont plus cohérentes avec l'architecture d'une application Angular. J'ai donc retenu l'utilisation du composant parent pour conserver l'état des filtres.

Implémentation des actions et de la donnée

J'ai ensuite implémenté le composant metadata qui est la colonne vertébrale de l'application. C'est lui qui est en charge de la récupération et des traitements des données pour ainsi les distribuer aux différents composants. Il fait le lien entre les recherches qui consomment les services externes, les filtres sur les résultats et l'affichage sur la carte. C'est également dans ce composant que l'implémentation du filtre « highlight » qui permet de mettre en surbrillance un élément sur la carte est réalisé. Il a été laborieux à mettre en place car il y a trois états de surbrillance à gérer qui sont liés entre eux comme suit :

1. L'état **normal** lorsqu'aucun élément n'est en surbrillance.
2. L'état de **surbrillance** lorsqu'on sélectionne un élément, tous les autres sont désactivés. De plus, la sélection est incrémentale
3. L'état **désactivé** lorsqu'un ou plusieurs éléments sont en surbrillance, les autres éléments ont l'état désactivé.

Implémentation des services externes

Une fois l'implémentation des vues et des filtres terminés, je suis passé à l'implémentation des services Angular afin de remplacer les mocks de données par la consommation de services externes. L'entreprise fait systématiquement un [wrapper](#) pour chaque service afin d'abstraire la couche HTTP. Le wrapper fournit des méthodes qui renvoient la donnée correspondante aux filtres. J'ai débuté par l'implémentation des méthodes du [wrapper SIG](#) permettant de requêter sur le serveur cartographique afin de récupérer la liste de types et d'effectuer la recherche d'objets contenus dans ces types.

Implémentation de la carte

Maintenant que les objets sont récupérés, il faut les afficher sur la carte.

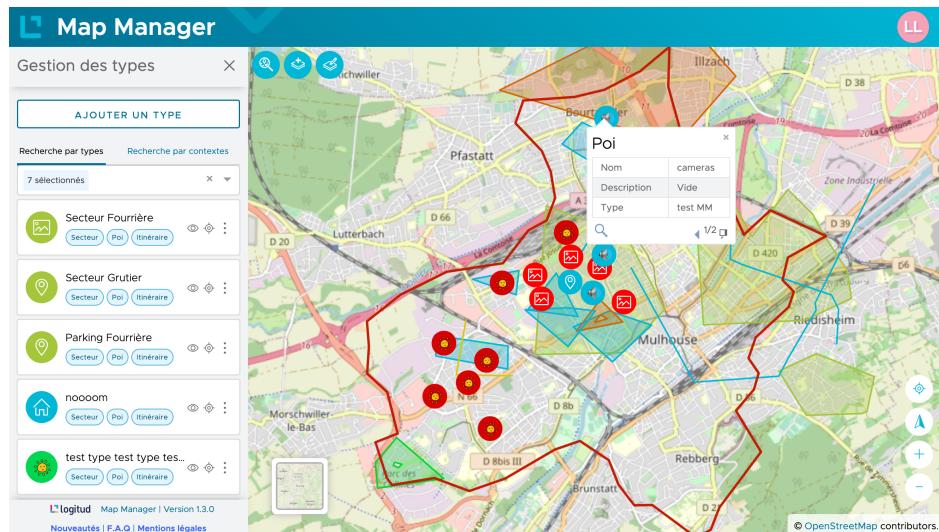


FIGURE 13 – Map Manager

L'application consomme le serveur GeoToolbox qui lui retourne des collections d'objets au format GeoJSON, cf. [annexe 1](#). J'ai utilisé le pattern *subject/subscriber* implémenté par d'Angular. Le composant metadata publie les objets récupérés par le wrapper et le composant de la carte les affiche si les conditions d'affichage sont remplies.

Pour afficher une source de données sur la carte dans ce format il suffit de les lire avec le « Reader » du Format GeoJSON fourni par la librairie Openlayers qui permet ensuite de les ajouter à une couche vectorielle qui elle-même est ajoutée à l'instance de la carte. Nous avons décidé de créer un service jouant le rôle d'adaptateur et contenant l'implémentation des méthodes de la librairie OpenLayers afin de simplifier le changement de librairie cartographique si elle ne répond plus à nos besoins. Il suffit alors d'appeler au service de la carte la méthode correspondante à l'action

à réaliser en lui passant l’instance de la carte et les paramètres attendus.

Maintenant que les objets sont ajoutés à la carte, il faut définir leur style. Les métadonnées des objets géographiques, comme évoqué lors de la présentation de la géomatique, contiennent des données sémiologiques, définissant en l’occurrence la couleur et le pictogramme de l’objet. Elles doivent être utilisées pour construire le style. Le serveur encode la couleur en hexadécimal qui est supporté par la bibliothèque. En revanche, le pictogramme provient des différentes bibliothèques FONTAWESOME, CLARITYICONS ou le service LABELS avec des icônes personnalisées par le client. La bibliothèque OpenLayers constraint à l’utilisation d’icônes encodées uniquement en base64 : `data:image/svg+xml;base64,...`. Dans le cas des icônes du service labels, c’est simple, on les récupère dans le bon format de données. En revanche, dans le cas des deux bibliothèques d’icônes, on récupère uniquement son nom. Il a donc fallu le récupérer au format SVG à partir de son nom, sur certains réaliser des traitements puis le convertir dans le format attendu. J’ai réalisé de nombreux essais pour récupérer l’icône de la bibliothèque CLARITYICONS, car en effet, c’est assez facile de récupérer le code SVG de l’icône, mais il faut le traiter par la suite :



FIGURE 14 – Icône « Clock » de Clarity non traité vers l’icône traité

Le SVG, est un format d’image basé sur le XML. Cela signifie que l’on peut le manipuler afin de supprimer les noeuds XML contenant les géométries normalement cachées par le style du document (CSS) qui n’est pas pris en compte par les canvas utilisés par OpenLayers pour afficher la carte ainsi que les icônes. Pour le reste, j’ai pu me servir du code de la version précédente afin d’afficher les objets sur la carte.

Aussi important que de pouvoir afficher les objets sur la carte, il faut pouvoir les créer en les dessinant et plus généralement, exécuter des actions à partir de la carte. C’est ce sur quoi j’ai ensuite travaillé. J’ai implémenté les « controls OpenLayers » permettant de réaliser des d’actions sur la carte et en dehors tels que le sélecteur de fonds cartographiques, les différents outils de dessin et de modification, le bouton pour ouvrir la barre latérale ainsi que les interactions de la carte (zoom, rotation, recentrage sur le contour de la ville).

L’enregistrement des dessins a été ardu. Un besoin exprimé était la possibilité de dessiner plusieurs géométries dans un seul objet géographique. Le GeoJSON le permet avec des poly-géométries, cependant OpenLayers produit des géométries simples. Il a fallut implémenter des méthodes de conversions de types de géométries de Point → MultiPoint, LineString → MultiLineString, Polygon → MultiPolygon. Je me suis intéressé ensuite à une demande d’évolution. L’application doit permettre la modification de géométries avec différents outils comme la mise à l’échelle, la rotation, le déplacement de coordonnées ou encore la suppression d’objet. J’ai fait cela à partir d’interactions déjà existantes dans une extension de la bibliothèque OL-EXT.

Cette évolution doit permettre également de modifier plusieurs objets à la fois et de types différents.

Import de géométries

La dernière fonctionnalité manquante dans l'application avant de pouvoir la mettre en production est l'import d'objets cartographiques. Deux comportement ont été demandés pour l'import :

- Importer des objets dans un type existant.
- Importer et fusionner les géométries des objets dans un objet existant.

J'ai fait le choix de prendre de la dette technique en introduisant de la complexité inutile afin de livrer une première version de l'application rapidement car la présentation et la mise en production devait être imminente.

Tout d'abord, j'ai créé un composant pour téléverser le fichier contentant les objets à importer par glissement (drop) avec des vérifications nécessaire pour s'assurer que le fichier pourra être traité par la suite (taille, format). J'ai ensuite créé un composant pour choisir la projection de la donnée. La correspondance des données attributaires et les géométries sont gérés par le composant d'import qui va sérialiser les données afin de les afficher sur la carte. OpenLayers met à disposition des développeurs des serialiseurs pour les formats GeoJSON et KML. Pour supporter l'import du format Shapefile qui est un standard de nombreux logiciels SIG, il a fallu passer par une librairie externe permettant de convertir auparavant les données en GeoJSON.

Ensuite, il faut traiter les métadonnées associés. Pour ça, un formulaire permet de définir la correspondance entre les métadonnées des objets importés et les métadonnées de notre système :

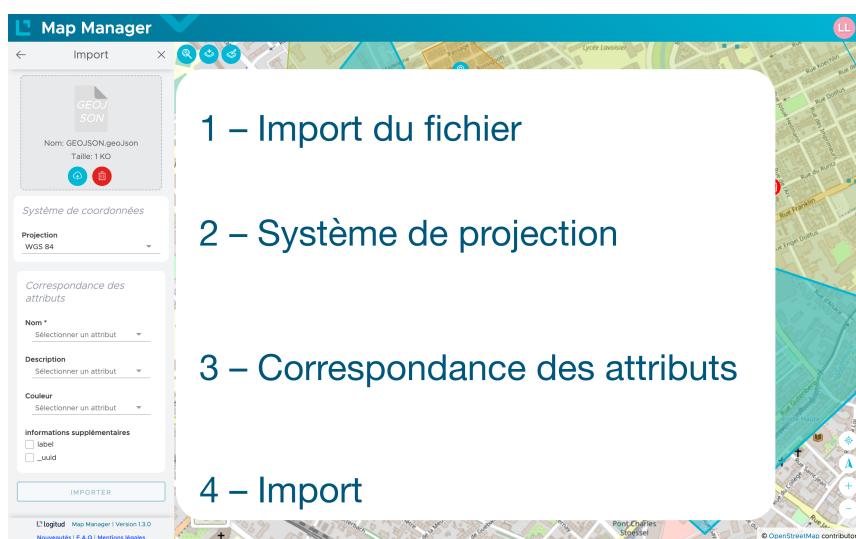


FIGURE 15 – Vue de l'import d'objets

En ce qui concerne l'interface utilisateur, la plus grande partie avait déjà été codé dans l'ancienne version de l'application. Il a été nécessaire de séparer l'enregistrement en fonction de la destination de l'import : dans de nouveaux objets à ajouter à un type existant ou bien fusionner un objet existant.

Documentation de l'application

La dernière étape avant de créer le premier tag de l'application a été de générer le changelog (cf. [annexe 3](#)), la documentation des changements du projets entre les différentes versions. J'ai depuis le début de la réécriture de l'application, fait le choix de suivre la convention de commit d'Angular connu pour rendre l'historique de versionnement explicite. Elle décrit explicitement le type de modification réalisée. Pour exemple avec l'ajout d'une fonctionnalité dans les objets géographique :

```
feat(geofeatures-component): add layer-cards component
```

La convention concorde avec la convention de versionnage des application **SemVer** qui est utilisée par l'entreprise, ayant trois chiffres : le premier pour les modifications cassantes, le deuxième pour les nouvelles fonctionnalités et le troisième pour les corrections. J'ai mis en place un script comparant les commits depuis le précédent tag de version afin de générer le changelog et de monter la version de l'application automatiquement. En fonction du type des commits (*feat, fix, perf, ci, refactor, docs, build*), la partie correspondante de la version est augmentée.

3.6 Déploiement

Pour tester l'application lors du développement, nous avons mis en place un environnement d'intégration continue afin de déployer régulièrement l'application sur l'environnement de test.

Suite à un non-versionnement de l'API du serveur cartographique GeoToolbox et un breaking change, Map-Manager a été déployé en production un peu plus rapidement que planifié initialement, en Mai. En effet suite à une demande d'évolution dans le serveur cartographique GeoToolbox pour le nouveau Map-Manager produisant un changement cassant (breaking-change) dans l'API du serveur backend GeoToolbox qui abstrait maintenant le type de géométrie des objets afin de pouvoir tous les résupérer à partir d'un seul end-point. L'ancienne version n'était plus fonctionnelle et à ce moment, toutes les anciennes fonctionnalités avait été implémentées dans la nouvelle version de l'application. Pour déployer l'application en production, il faut ouvrir une demande sur le site du support pour que le service de déploiement puisse déployer l'application pour tous les clients.

3.7 Améliorations et perspectives

Bien que l'application contienne plus de fonctionnalités que la précédante, plusieurs évolutions sont encore à implémenter et d'autres envisagées.

Permettre à l'utilisateur une personnalisation de son application en lui permettant de sauvegarder des préférences d'affichage et également de permettre au client d'ajouter son flux cartographique personnel pour les fonds cartographiques.

De nouvelles fonctionnalités sont également planifiées dans la prochaine version. L'utilisation d'un serveur de moteur de rendu cartographique permettant de générer la carte affiché dans un fichier à partir d'un template défini est actuellement en cours d'implémentation dans la librairie map-viewer et sera également implémentée dans map-manager.

De plus, également de nouvelles évolutions pas encore programmées ont été annoncées. Il serait également intéressant d'implémenter le visualiseur d'image [Mappillary](#) permettant de visualiser la rue sur des photos partagées par des contributeurs et de pouvoir visualiser et dessiner des objets géographiques à l'intérieur.

Glossaire

agile La méthodologie Agile est un modèle d'organisation de projet qui place le client à son cœur. [2](#)

App Shell Squelette de l'application. [ii](#), [10](#)

applications métier Applications destinées à l'utilisateur final pour répondre à son besoin issu de son métier. [4](#), [6](#), [19](#)

framework SCRUM Lié à la méthodologie Agile. Il améliore la productivité des équipes tout en permettant une optimisation du produit grâce au retours des clients. [2](#)

proxy Hôte intermédiaire se plaçant entre deux hôtes pour faciliter ou surveiller leurs échanges. [6](#)

reverse-geocoding Attribution d'une adresse à partir de coordonnées géographique. [6](#)

SIG Système d'information géographique. [ii](#), [1](#), [3](#), [4](#), [6](#), [8](#), [13](#), [15](#), [19](#)

wrapper un wrapper est une entité qui encapsule et masque la complexité sous-jacente d'une autre entité au moyen d'interfaces bien définies. Dans le cas d'un wrapper HTTP, il encapsule et masque la couche HTTP et renvoie les données après traitement ou brut.. [13](#)

Références

[1] IGN 2008, *Le repère RGF93 et la projection Lambert-93*, <http://aiweb.techfak.uni-bielefeld.de/content/bworld-robot-control-software/>

Conclusion

À mon arrivée dans le service, mon collègue m'a confié de diverses tâches dans les différents miniservices cartographiques, me permettant ainsi de découvrir leurs différentes fonctionnalités. L'absence de documentation m'a permis de m'extraire et d'échanger avec mes collègues. Elle m'a également permis d'apprendre l'origine des choix et décisions techniques réalisés, d'enrichir mes connaissances sur le domaine métier et le besoin des clients. J'ai également eu l'occasion d'amener l'utilisation de certaines bonnes pratiques permettant de gagner du temps et d'augmenter la maintenabilité. Le projet Map-Manager dont j'ai eu la tâche de refonte et d'ajout de nouvelles fonctionnalités, exploité par les différentes **applications métier**, m'a également permis de découvrir l'application de la gamme sécurité destinée à la police municipale, « MunicipolWEB 2 », sur laquelle portera ma prochaine mission avec l'enchaînement sur un contrat de professionnalisation.

Mots clés

- Refonte
- Cartographie – **SIG**
- Framework Web – Angular

ANNEXE

ANNEXE 1 – Exemple d'une collection d'objets contenant un POI au format GeoJSON

```
1  {
2    "features": [
3      {
4        "geometry": {
5          "coordinates": [
6            7.3306349332097,
7            47.75105398476878
8          ],
9          "type": "Point"
10        },
11        "type": "Feature",
12        "properties": {
13          "color": "#01B7D6",
14          "provider": "CUST",
15          "name": "camera",
16          "description": "",
17          "id": "CAMERA_1653034125850_1",
18          "type": {
19            "icon": {
20              "value": "ICONE/DEFAUT/AMPOULEA",
21              "source": "LABELS",
22            },
23            "id": "STATIONNEMENT_1638527874599_8",
24            "name": "Stationnement",
25            "color": "#064bf3",
26            "provider": "CUST",
27            "isFavorite": false,
28            "contexts": []
29          },
30          "category": "POI",
31        }
32      },
33    ],
34    "type": "FeatureCollection"
35 }
```

ANNEXE 2 – Document réalisé rendant compte des fonctionnalités de la version avant refonte

Analyse et définition des fonctionnalités de Map Manager

Cette application permet d'éditer des zones, des POI (points d'intérêts) ainsi que des itinéraires séparément. => linestring, polygon et point sont séparés

Lorsqu'on arrive sur l'application, on doit alors choisir entre ces trois:



ZONES

POI

ITINÉRAIRES

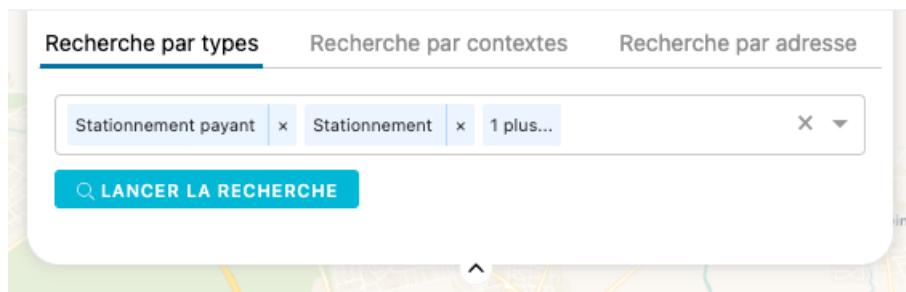
Fonctionnalité de recherche

La fonctionnalité de recherche permet d'afficher les couches ainsi que tous les objets qu'elle contient. Pour afficher les informations contenues dans les types, il faut utiliser la fonctionnalité de recherche par type.

La recherche par contexte permet d'afficher tous les types qui sont liés au/aux contextes.

La fonctionnalité de recherche par adresse n'est pas entièrement implémentée. Elle est censée filtrer dans toutes les couches, les features qui touchent le point. Pour l'instant, elle affiche un point (comme les POI) aux coordonnées de l'adresse.

Pour l'instant, il n'y a pas de filtres sur les features.



Paramètres (Gestion des types)

Un type est une couche. Il a un nom défini lors de la création, une couleur, une icône, un ou des contextes.

Un type est défini par un nom, une couleur, une icône et un/des contextes métiers (optionnel)

L'icône (POI) et la couleur définis sont utilisés comme style de base pour les features (s'il ne sont pas surchargé dans la feature).

	Nom	Couleur	Icone	Contextes
+	Stationnement payant	■		PreControls FPS
+	Stationnement	■		STATIONNEMENT Main courante Opération tranquille Verbalisation élémentaire Fourrière véhiculaire Arrêtés

Ajouter une feature (un POI, une zone, un itinéraire)

Il faut ouvrir le tableau “Liste de données” en bas de page. Il s’ouvre également automatiquement lorsqu’on lance une recherche. Puis cliquer sur “Ajouter”:

Liste de données

+ AJOUTER

Nom

Une modal permettant d’ajouter une feature s’ouvre:

Ajout d'une zone

Type *	Veuillez choisir un Type		
Parent	Veuillez choisir un parent		
Nom *	Veuillez saisir un nom	Couleur *	<input type="checkbox"/>
Adresse	Veuillez saisir une adresse		
Description	Veuillez saisir une description		
<input type="button" value="+ AJOUTER UNE INFO"/>			
		<input type="button" value="ANNULER"/>	<input type="button" value="VALIDER"/>

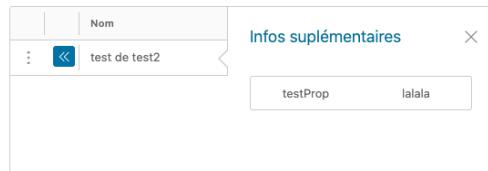
Elle permet de créer une zone:

- de lier le type à la feature
- de faire hériter une forme depuis un parent (une zone dans une zone)
- Donner un nom à la zone
- Une couleur à une zone
- Une adresse (non utilisé par les filtres)
- Description

Il est également possible d’ajouter des propriétés personnalisées.

Propriété *	Valeur *	<input type="button" value=""/>
<input type="button" value="+ AJOUTER UNE INFO"/>		

elles seront affichées dans le tableau



Une fois la feature créée, on peut dessiner la géométrie en cliquant sur le crayon:



Dessiner la ou les zones / POI / Itinéraires:



Puis enregistrer la forme géométrique



Fonctionnalité d'import



L'application permet d'importer un itinéraire, POI ou zone depuis l'onglet import

Après avoir sélectionné un fichier KML ou geojson:

The screenshot shows a map application interface. On the left, there is a configuration panel for a KML layer named "m2a_communes-sur-le-territoire-m2a.geojson" (Size: 917 KO). The panel includes settings for coordinate systems (Projection: WGS 84 / Pseudo-Mercator par défaut) and attribute mapping (Type, Nom, Description, Couleur). Below these are checkboxes for additional information: popu_munic, com_nom, and canton. On the right, a map of the Alsace region is displayed, showing various towns and administrative boundaries.

On peut / doit:

- sélectionner la projection à partir d'une liste définie
- Sélectionner le type dans lequel importer la couche
- Binder les attributs de la couche avec les attributs de l'application (nom, description et couleur)

Map tools



Les outils de la carte se trouvent en haut à droite de l'application.
L'outil layers permet de choisir le fond de carte. L'outil plein-écran permet de cacher l'app-shell. L'outil dessiner permet également de dessiner une forme si une donnée est sélectionnée.

Issues rencontrées:

- Lorsqu'on ajoute plusieurs features dans la collection, seule la dernière dessinée est prise en compte lorsqu'on clique dessus dans le tableau (avant de recharger la page).
- Idem avec les props custom: elles ne sont pas affichées lorsqu'on en ajoute depuis le tableau (avant de recharger la page) et sont écrasées lorsqu'on en ajoute une autre depuis la map.
- Import: L'import d'un fichier geojson (certainement hors limites) s'est importé la première fois en double. Il a été impossible d'en ré-importer un (le même ou un autre avec une zone contenue dans la BBOX limite) dans n'importe quel groupe. On a comme erreur "HTTP status code 500" (au lieu 400, vu le détail de l'erreur), "code: FORMAT_ERROR" et dans le détail, on retrouve "code: Validation Error" et "message: Géométrie en dehors de la commune".
- Champ de recherche vide non validé dans le back

ANNEXE 3 – Changelog généré automatiquement pour la version 1.2.1

Changelog

All notable changes to this project will be documented in this file. See [standard-version](#) for commit guidelines.

1.2.1 (2022-06-02)

Features

- add padding to fit extent function ([620bef0](#))
- add spinner to map ([07d4af5](#))
- disable highlight on outside click ([8321dbe](#))
- handle multi customer cities and load them after initialization ([75a7af2](#))

Bug Fixes

- **icon-picker:** provide default labels root sites ([f5bd091](#))