



---

# Refonte et amélioration d'une application de cartographie SIG

---

## Rapport de stage

ROBIN RULLO

LICENCE PROFESSIONNELLE  
DÉVELOPPEUR INFORMATIQUE

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 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 dans le service, 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 à mes questions et avec qui j'ai pu échanger des connaissances pour ainsi progresser.

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Organisme d'accueil</b>	<b>2</b>
<b>3</b>	<b>Le stage</b>	<b>3</b>
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 .	4
3.3.2	État de l'art . . . . .	4
3.4	Définition du besoin . . . . .	6
3.5	Développement du projet et difficultés rencontrées . . . . .	6
3.5.1	Maquettage . . . . .	6
3.5.2	Implémentation de la maquette . . . . .	7
3.6	Déploiement en production . . . . .	13
3.7	Améliorations et perspectives . . . . .	14
<b>A</b>	<b>ANNEXE</b>	<b>16</b>

## Table des figures

1	Les métiers couverts par les suites logicielles de Logitud Solutions . . . . .	2
2	Choix de la vue selon le type de géométrie . . . . .	5
3	Place monopolisée par les menus . . . . .	5
4	Première proposition . . . . .	6
5	Proposition retenue . . . . .	7
6	Carte d'un type et en-dessous celle d'un secteur . . . . .	9
7	Composant de recherche de types à gauche et composant de visualisation du contenu du type à droite . . . . .	9
8	Icon « Clock » de Clarity non traité vers l'icon traité . . . . .	11
9	Vue de l'import d'objets . . . . .	12
10	Architecture SIG . . . . .	16

## 1 Introduction

Logitud Solutions SAS 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. Afin de rester leader du marché, elle doit faire face aux logiciels de la concurrence et est forcée de se maintenir à jour technologiquement. Elle a amorcé depuis quelques années la réécriture de ses applications lourdes, nécessitant une installation sur un poste de travail, vers des applications web fonctionnant sur un navigateur web. Ces applications traitent un lot données géo-référencées, lesquelles nécessitant quelques fois des traitements.

M'étant spécialisé dans les systèmes d'information géographique depuis le début de la formation à l'UHA 4.0, je souhaitais, en déposant ma candidature, assoir 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. En fin, une conclusion boucle le rapport.

L'application web SIG permettant de gérer les contextes géographiques de la suite d'applications n'est plus adapté aux nouveaux besoin et n'est pas suffisamment robuste pour être déployé en production.

## 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 compte 90 employés dont 30 développeurs. Elle est spécialisée dans l'édition d'outils numériques destinés aux collectivités locales (communes, communautés 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.



FIGURE 1 – Les métiers couverts par les suites logicielles de Logitud Solutions

L'entreprise est aujourd'hui un acteur majeur du marché des logiciels à usage des collectivités territoriales, équipant un tiers des villes de plus de 5 000 habitants en produit d'état-civil, et les quatre cinquième en produit de sécurité (polices municipales).

La méthodologie de travail adopté dans les équipes de développement suit indépendamment les principes de la méthodologie SCRUM. L'équipe de R&D dont est rattaché le pôle de développement cartographique bénéficie d'un peu plus de liberté dans son management.

## 3 Le stage

### 3.1 Présentation du contexte

Afin de s'adapter aux services proposés par la concurrence, 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 dans la diffusion des données.

Afin de se reprérer et de localiser l'information sur la surface terrestre, il est nécessaire d'utiliser un système de position 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, nous utilisons le RGF (Réseau Géodésique Français) 93. Cependant, la norme pour certain format de données est le WGS (World Geodetic System) 84, utilisé par les américains et associé au système GPS.

**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.

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 (taille, couleur, pictogrammes, etc...).

### 3.3 Étude de l'existant

A mon arrivée, MAP MANAGER avait déjà subit trois refontes. Plusieurs problématiques qui n'avaient pas été posées au début du développement et qui se sont ajoutés dans le temps ont rendu l'application obsolète.

Map-Manager est Single Page Web Application (SPA – application web à page unique). Les autres applications, consommant des données géographiques, il a fallut créer une librairie Angular, baptisé Map-Viewer, 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 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 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 icone. 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 au module d'une application permettant d'assigner le type au domaine métier d'une application quelconque. Les contextes métiers sont gérés par le service **LABELS** qui est commun à toutes les applications.

#### 3.3.2 État de l'art

L'application permettait l'administration de secteurs, itinéraires et poi sur trois pages différentes.



FIGURE 2 – Choix de la vue selon le type de géométrie

Une fois une vue sélectionnée, il est possible de sélectionner un ou plusieurs types soit contextes. Des géométries s'affichent et un tableau souvre sur la moitié horizontale basse de l'écran, affichant les géométries correspondants aux critères de recherche. Sur de petits écrans, la place est monopolisée par le menu de recherche et le tableau listant les objets géographiques :

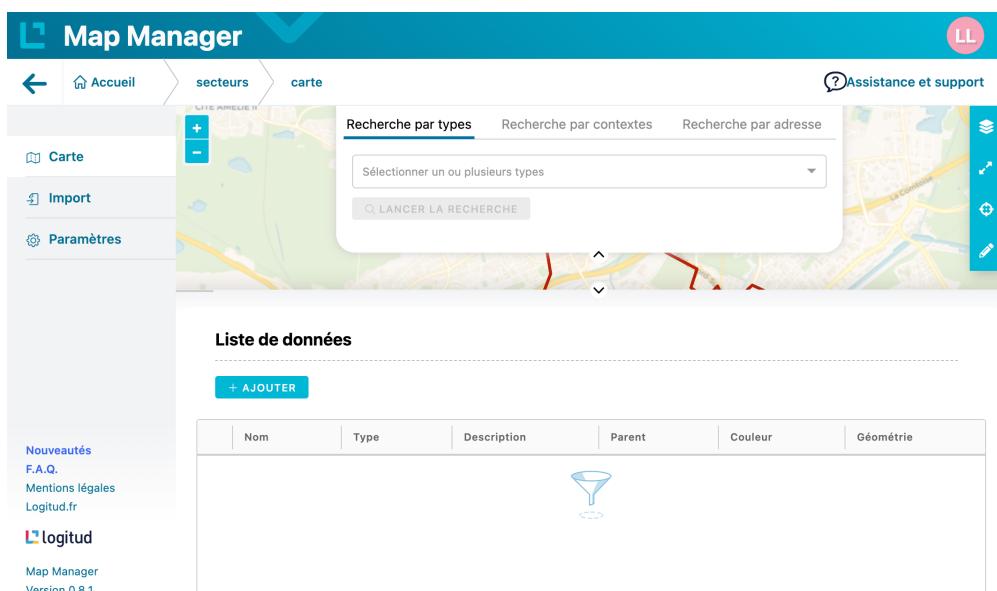


FIGURE 3 – Place monopolisée par les menus

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

Une page paramètre dans l'application permet de réaliser un CRUD 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 intéragir avec les objets géographiques, il faut passer par le serveur Geo-Toolbox. 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 géomaticien.

### 3.4 Définition du besoin

1. L'application doit être iso-fonctionnelle.
2. Afficher toutes les catégories d'objets géographique sur une seule carte.
3. Application mono-page carto-centré
4. L'application devra utiliser les librairies communes aux autres applications.

### 3.5 Développement du projet et difficultés rencontrées

#### 3.5.1 Maquettage

J'ai commencé par réaliser une maquette graphique. J'ai alors fait deux propositions. Nous avons organisé une réunion avec le référent UI/UX, le responsable du pole R&D, le responsable des développements et également le responsable de la hotline afin de valider l'implémentation du besion métier et l'ergonomie à l'utilisation.

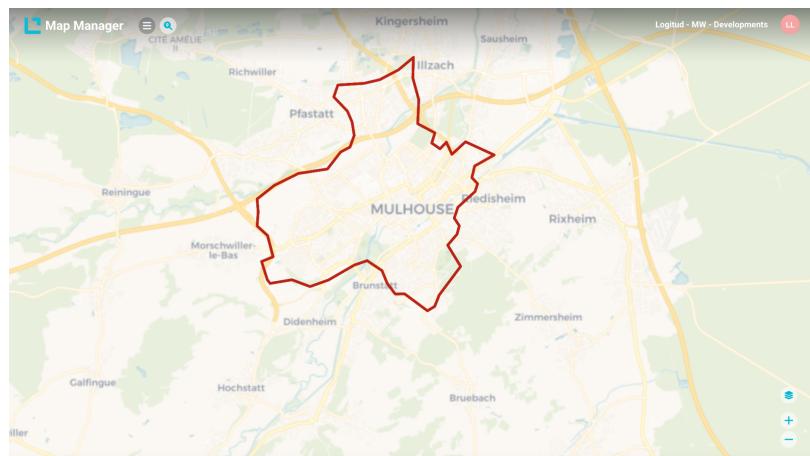


FIGURE 4 – Première proposition

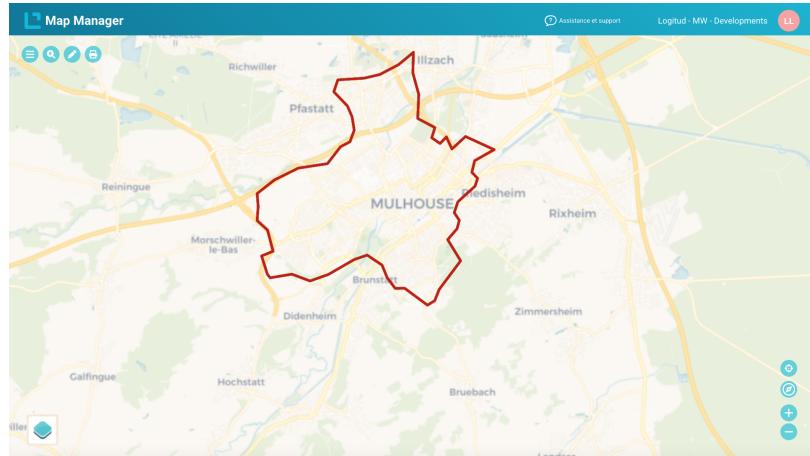


FIGURE 5 – Proposition retenue

### 3.5.2 Implémentation de la maquette

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 aux applications de l'entreprise sont en version 9 de trop nombreux breaking-changes serait à corriger.

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
```

J'ai ensuite commencé le développement des fonctionnalités. Nous verrons d'abord la définition des models et interfaces, puis la création des composants de l'interface utilisateur. Nous verrons ensuite l'interaction des données avec les autres services de la suite logicielle. Nous analyserons l'implémentation de la carte avec la génération du style puis les interactions avec celle-ci. Nous verrons par la suite la fonctionnalité d'import d'objets géographiques. Nous nous intéresserons également à la librairie map-viewer, au service d'impression et la documentation du projet.

Dans l'entreprise, tous les services possédant une API possèdent également un Wrapper. Il permet d'abstraire et faciliter les requêtes vers son service en appelant sa méthode correspondante au besoin. Malheureusement, les wrappers ne sont pas uniformisés donc chaque wrapper à une implémentation différente et ces interfaces ne sont pas toujours à jour avec celles du service.

Map-Manager intérage avec plusieurs services. Le choix a été de se focaliser de prime abord sur l'intégration de la maquette dans angular. C'est pourquoi toutes les interfaces ont été redéfinis et les services externes ont consommés des mocks de ses services.

Toujours dans le but de faciliter les développements et d'acroître la maintenabilité, dans la partie front-end l'entreprise maintient une librairie de composant Angular, baptisé WebUI-core, évitant d'une part la répétition de code et le développement de composant déjà existant dans d'autres applications et d'autre part elle permet d'uniformiser l'interface des différentes applications et l'évolution des composants dans toutes les applications peut se faire simplement en montant la version de la librairie. D'ailleurs, la librairie contient également un composant Map-Viewer, que l'on verra plus en détail par la suite, reprennant certaines fonctionnalités de Map-Manager.

Map-Manager étant une application carto-centré détaché du métier des autres applications, il a fallut repartir de zéro pour implémenter le design de la maquette car ce concept n'est implémenté nul part ailleur dans les autres applications et donc aucun composant n'est intégré à la librairie WebUI-core. J'ai alors commencé par implémenter les cartes d'information des objets géographiques et de leurs types : Puis j'ai continué par implémenter le composant d'affichage des objets géographiques et le composant de recherche de types avec leurs recherche : . J'ai rencontré quelques difficultés lors de l'implémentation. Le changement de vue dans la sidebar vers le composant d'affichage des objets géographiques provoque la destruction (dans les cycles de vie des composants Angular) du composant d'affichage des types, donc tous les filtres de recherches sont perdus. Cela m'a rappelé la problématique dans laquelle était facebook il y a quelques années et dont découle l'architecture basée



FIGURE 6 – Carte d'un type et en-dessous celle d'un secteur

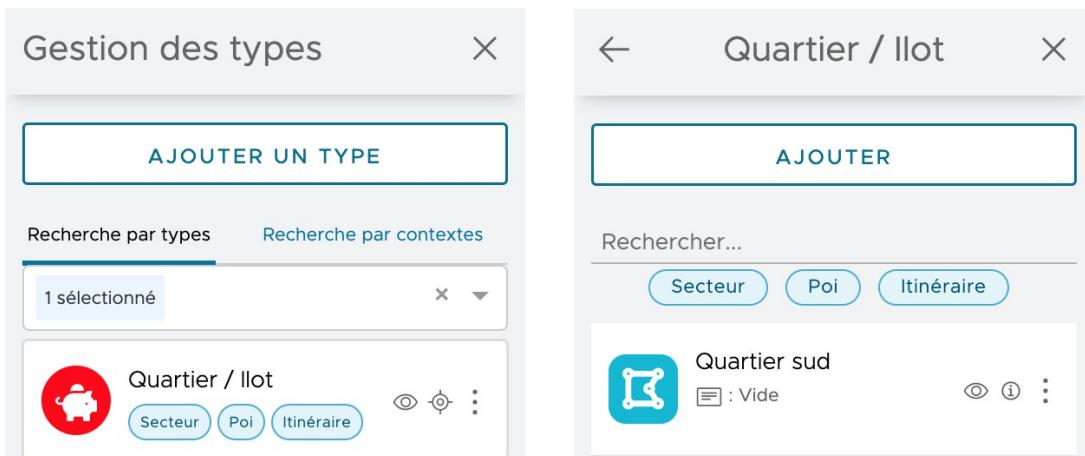


FIGURE 7 – Composant de recherche de types à gauche et composant de visualisation du contenu du type à droite

sur les flux. L'état aurait été gardé dans un store global et le composant mis à jour lorsque cet état aurait été modifié. Je voulais alors mettre en place un store dans l'application, cependant j'étais le seul à être à l'aise avec une architecture basé sur les flux dans l'entreprise et stocker l'état dans le composant parent ou dans un service était plus cohérent dans l'architecture d'un projets Angular. J'ai donc utilisé le composant parent pour conserver l'état des recherches et filtres.

Le composant metadata est la colonne vertébrale de l'application. C'est lui qui va être en charge de la récupération et des traitements des données. Il fait le lien entre les recherches (appels API), les filtres sur les résultats et l'affichage sur la carte. L'implémentation du filtre highlight qui permet de mettre en surbrillance un élément sur la carte a été une difficulté, car il y a trois états à gérer et 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.

Une fois l'implémentation des vues et des filtres terminés, j'ai remplacé les mocks de données par l'implémentation des services externes. J'ai débuté par l'implémentation des méthodes du GIS-Wrapper permettant de requêter sur le serveur cartographique. Cela afin de récupérer la liste de types et d'effectuer la recherche d'objets contenus dans les types.

Suite à la refactorisation du serveur ayant pour but d'unifier les secteurs, les POI et les itinéraires en une seule entité avec un discriminant pour la catégorie, j'ai également dû mettre à jour le GIS-Wrapper en implémentant de nouvelles méthodes plus génériques et en déprécient les anciennes que j'ai dû proxier sur les nouvelles pour ne pas créer de Breaking-Changes. J'ai également implémenté le wrapper du repository, permettant de récupérer les informations sur l'utilisateur connecté ainsi que le wrapper pour le service label afin de récupérer les contextes affectés aux types ceux-ci permettant de compartimenter et filtrer les types disponibles dans les modules des applications métier.

Maintenant que les objets peuvent être récupérés, il faut les afficher sur la carte. Pour afficher une source GeoJSON qui le format des objects géographieus récupérés depuis le serveur sur la carte, il suffit de les parser avec le Reader du Format GeoJSON fournit par Openlayers qui permet ensuite de les ajouter à une couche vectorielle qui elle même est ajouté 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 allors d'appeler au service de la carte la méthode correspondante à l'action a réaliser en lui passant l'instance de la carte et les paramètres attendus.

Pour ajouter les objets récupérés par le composant metadata, nous avons suivi le pattern de subject/subscriber que conseille d'utiliser Angular afin d'émettre les nouveaux objets à chaque changements et de les mettre à jour sur la carte. Au moment de les ajouter sur la carte, il faut vérifier que l'objet ainsi que son type soit bien activés ainsi que de définir un style pour l'affichage, une autre difficulté.

Les metadata des objets géographiques, comme évoqué lors de la présentation de ce qu'est la géomatique, contiennent des données sémiologiques, définissant la couleur et le pictogramme de l'objet. Elles contiennent également d'autres données, métier, provenant des applications. Par exemple, les metadata d'un secteur dans le type stationnement payant pourra contenir le prix du stationnement en fonction des

horaires. Il faut exploiter les données de metadata afin de construire le style pour l'affichage des objets. Le style est défini par une couleur et un pictogramme. La couleur est en hexadécimal et le pictogramme est un pictogramme des différentes librairies FontAwesome, ClarityIcons ou le service Labels de l'entreprise. Dans le cas des labels, c'est assez simple, on récupère l'icon dans le bon format de données. En revanche, dans le cas des deux librairies d'icons, on récupère uniquement le nom de l'icon. La librairie cartographique ne permet uniquement l'utilisation d'icons sous forme de données encodés en base64 : `data:image/svg+xml;base64,...`. Il a donc fallu récupérer l'icone au format SVG, le traiter et le convertir. SVG pour « Scalable Vector Graphics » est un format d'image basé sur le XML. J'ai réalisé de nombreux essais pour récupérer l'icone de la librairie ClarityIcons, car en effet, c'est assez facile de récupérer le SVG de l'icon mais il faut le traiter afin de supprimer les noeuds contenant les paths cachés par défaut 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 icons :



FIGURE 8 – Icon « Clock » de Clarity non traité vers l'icon traité

Pour afficher des objets géographiques, il faut pouvoir en dessiner. C'est ce sur quoi j'ai ensuite travaillé. Il a fallut jouer avec les conversions de types de géométries de Point → MultiPoint, LineString → MultiLineString, Polygon → MultiPolygon afin de pouvoir dessiner plusieurs géométries (d'un seul type) dans un seul objet. Un fois fait, il a également fallu gérer la modification avec différents outils comme la mise à l'échelle, la rotation, le déplacement de coordonnées ou encore la suppression d'objet à partir d'interactions déjà existantes proposées par la librairie cartographique. Il a fallut prendre en compte une demande assez importante dans l'implémentation de ces outils, c'est qu'il doit être possible de modifier plusieurs objets à la fois et de types différents.

Il a également fallut gérer les comportements utilisateurs et l'empêcher de changer de mode d'édition lorsqu'il n'a pas enregistré son travail.

La dernière fonctionnalité manquante de l'application avant de pouvoir la mettre en production est l'import d'objets cartographiques. Il faut pouvoir gérer deux comportement pour l'import :

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

Par manque de temps, j'ai commencé par réutiliser les composants développés dans l'ancienne version de l'application en les adaptant à la nouvelle architecture. J'ai créer un composant pour téléverser le fichier par glissement avec des vérifications nécessaire pour s'assurer que le fichier pourra être traité (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 des logiciels SIG, il a fallut passer par une librairie externe permettant de transformer auparavant les données en GeoJSON. Une fois les objets traités, il faut traiter les métadonnées associés. On a créer un formulaire afin de réaliser la correspondance entre les métadonnées des objets importés et les métadonnées de notre système : Pour l'interface utilisateur,

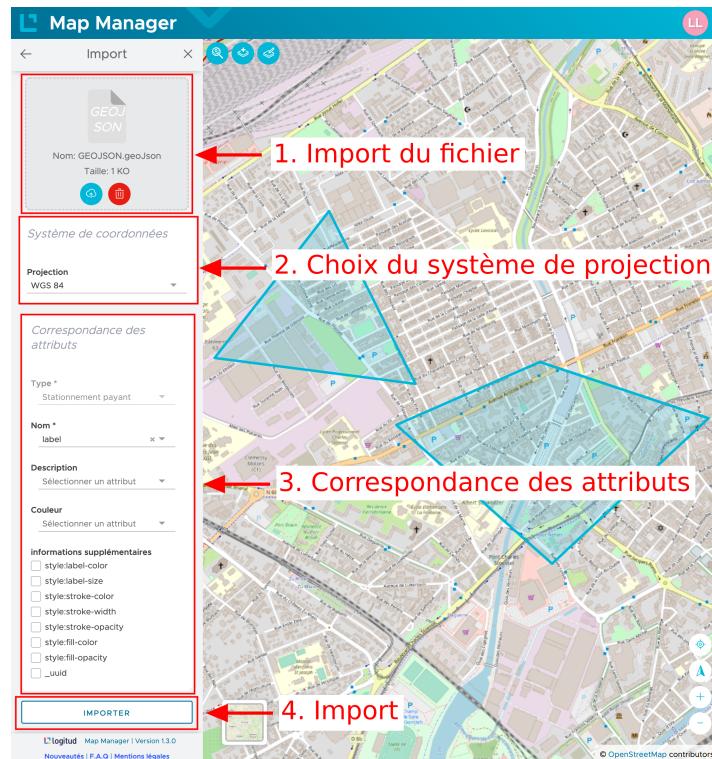


FIGURE 9 – Vue de l'import d'objets

la plus grande partie avait déjà été codé dans l'ancienne version de l'application. Il a uniquement fallut adapter les champs lorsqu'on import dans un type ou dans un objet existant.

J'ai commencé par définir le model des données et créer des mocks (données statiques utilisées pour les tests, imitant les données réelles), implémenter la partie shell (partie UI/UX fixe d'un site) puis j'ai créé les composants de l'interface à partir de la maquette. Nous avons fait le choix d'utiliser le système de design Clarity UI de VMWare car la librairie de composant partagée de l'entreprise, WebUI-core repose dessus. C'était assez compliqué d'appréhender ce système de design dont la version n'est plus maintenue.

J'ai ensuite implémenté la carte avec la librairie OpenLayers. J'ai d'abord créé des controls OpenLayers à partir de composants angular permettant de réaliser toute sortes d'actions sur la carte et en dehors. C'est le cas du sélecteur de fonds cartographiques, les différents outils de dessin et de modification, le bouton pour ouvrir la barre latérale, les interactions de la carte (zoom, rotation, recentrage sur le contour de la ville).

Après avoir implémenté les interactions sur la carte, j'ai ajouté les différentes couches contenant les objets de données. Il a fallu ajouter une couche pour les objets géographiques fournis par GeoToolbox, la couche pour les objets créé par les outils de dessin,

Pour déployer l'application sur l'environnement de test et de préproduction, nous avons mis en place une pipeline GitLab-CI afin de construire l'image docker et la déployer sur les serveurs.

La dernière étape avant de créer le premier tag de l'application a été de générer le changelog, 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. Exemple pour l'ajout d'une fonctionnalité dans les objets géographique : `feat(geofeatures-component): add` La convention concorde avec la convention de versionnage des application "SemVer" qui utilisée par l'entreprise, ayant trois chiffre : le premier pour les modifications cassantes (breaking-changes), 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 ainsi et 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 en production

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.

### 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.

Il serait intéressant de prendre en compte l'aspect de personnalisation en permettant à l'utilisateur de sauvegarder des préférences d'affichage et de permettre au client d'ajouter des fonds cartographiques personnalisés.

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 directement dedans.

## Résumé

L'absence de documentation des différents mini-services de la cartographie et des autres services en général a constitué un frein à mon arrivé, qui s'est relâché lorsque que mon collègue, dès le premier jour, m'a confié de diverses tâches me permettant de découvrir les différentes fonctionnalités des différents services cartographiques. Cette absence m'a également permis de m'extravertir et d'échanger avec mes collègues, et m'a permis également d'apprendre l'origine des choix et décisions techniques, d'enrichir mes connaissance sur le domaine métier et le besoin des clients. J'ai également eu l'occasion d'amener à l'utilisation de certaines bonnes pratiques permettant de gagner du temps et d'augmenter la maintenabilité. Le projet, 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'enchainement sur un contrat de professionnalisation.

(142w. max :150w)

## Mots clés

- Refonte
- Cartographie – SIG
- Framework Web – Angular

## A ANNEXE

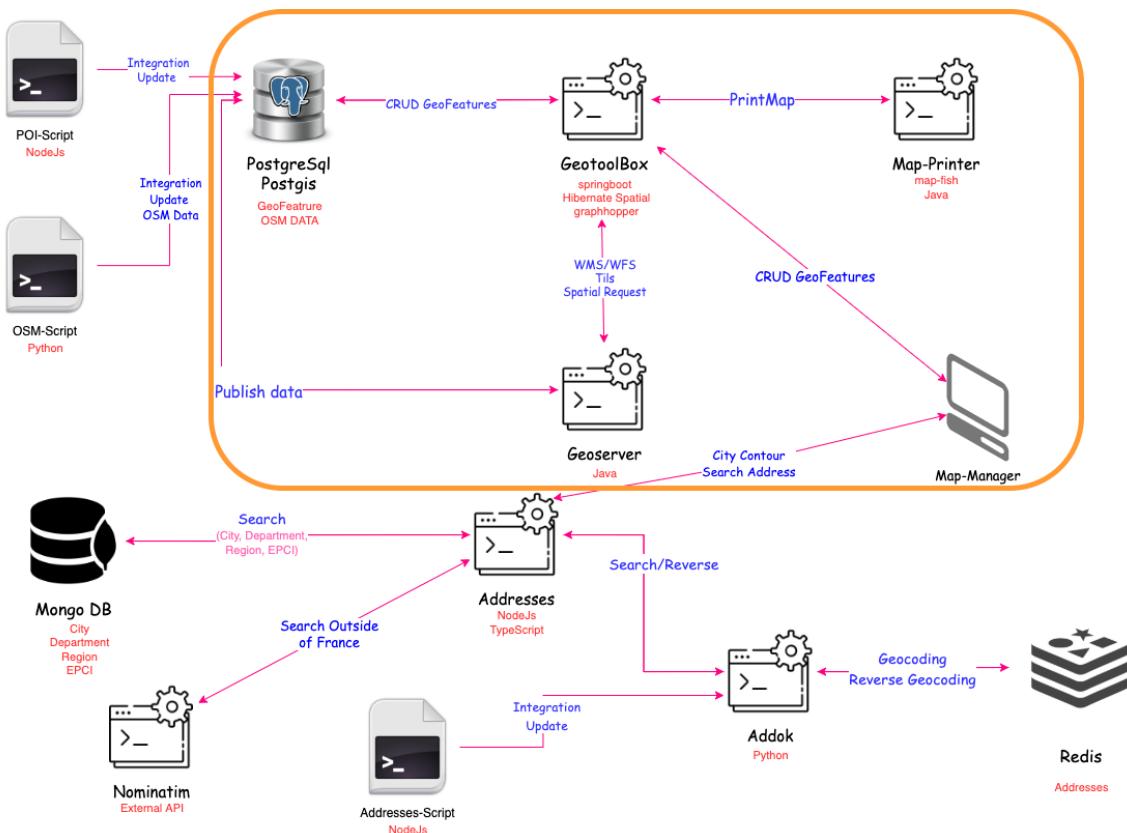


FIGURE 10 – Architecture SIG