Réalisé par Yann Bodiguel

Rapport de stage chez Geomatys

Implémentation d'un nouveau moteur cartographique dans une application web avec Angular

Tuteur universitaire : Anne Chirouze
Tuteur de stage : Dorian Ginane

BUT Informatique
Réalisation d'Applications : Conception, Développement, Validation (RACDV)

Année universitaire 2024-2025 du 3 février au 28 mars 2025

1. Remerciements :

La première personne que je tiens à remercier est Alessandro Valeri, qui s'est montré très disponible et patient à chacun des nombreux problèmes que j'ai rencontrés, ce qui a je pense contribué à ce que je n'hésite jamais à demander de l'aide, rendant très agréable l'expérience du stage.

Merci à Guilhem Legal pour m'avoir guidé dans la mise en place de mon projet lors des premiers jours,
À Dorian Ginane, mon tuteur de stage, Séverin Pâques, en charge de mon projet, Isabelle Pélissier, responsable administrative, financière et des ressources humaines, et Vincent Heurteaux, directeur général de Geomatys, pour leur suivi de mon avancé dans le projet, et du bon déroulement de mon stage.

De manière générale, merci à tous les gens présents dans l'entreprise pour avoir participé à me fournir de très bonnes conditions de travail à tous les niveaux.

2. Résumé :

Ce rapport de stage présente le travail que j'ai réalisé au sein de l'entreprise Geomatys, du 3 février au 28 mars 2025, en tant que développeur full stack, dans le cadre de ma deuxième année du BUT Informatique, plus précisément dans le parcours Réalisation d'Applications : Conception, Développement, Validation (RACDV) à l'IUT de Montpellier.

Sommaire :

1. Remerciements : 1
2. Résumé : 2
Sommaire : 3
3. Table des figures : 4
4. Glossaire : 5
5. Introduction : 6
5.1 Explication de la mission du stage : 7
5.2 Analyse technique : 9
5.3 Cahier des charges : 10
6. Rapport technique : 12
6.1 Je ne connais pas Angular 12
6.2 Lancement de l'application 12
6.4 Recréation de l'existant : 14
6.5 Sélection des capteurs : 15
6.6 Clustering : 17
6.7 Le capteur sélectionné qui disparaît avec le zoom : 19
6.8 Les capteurs qui se chevauchent : 21
6.9 Filtrer les layers : 22
6.10 Layers WMS customs : 23
6.11 Styles de la carte : 26
6.12 Fenêtre d'informations : 28
6.13 Importants problèmes de performance : 29
7. Méthodologie Organisationnelle : 33
7.1 Organisation du travail dans l'entreprise : 33
7.2 Versioning avec Git : 33
8. Conclusion : 35
8.1 Bilan du travail réalisé : 35
8.2 Bilan des apprentissages : 35
8.3 Signatures : 36
9. Bibliographie 37
10. Annexes : 38
10.1 Fiche technique du produit Aqualit : 38

3. Table des figures :

Fig.1 : Tableau du personnel de l'entreprise..... 6
Fig.2 : Présentation d'Aqualit..... 7
Diagramme de séquence du fonctionnement d'une partie de l'application..... 10
Fig.3 : Version améliorée de la Carte..... 16
Fig.4 : Capteur sélectionné avec Popup..... 17
Fig.5 : Comparaison avec et sans clustering..... 19
Fig.6 : 2 capteurs l'un au dessus de l'autre.....22
Fig.7 : Capteurs espacés..... 22
Fig.8 : Menu de filtrage.....23
Fig.9 : Une des légendes de la carte.....24
Fig.9 : Menu d'ajout de calques customs.....24
Fig.10 : Array de MapLayer, affiché dans la console de Mozilla Firefox..... 25
Fig.11 : Fenêtre d'information.....29
Fig.12 : Offset de l'icône sélectionné par rapport l'icône non-sélectionnée.....30

4. Glossaire :

Front end : partie d'une application ou d'un site web avec laquelle l'utilisateur interagit directement.

Back end : partie "cachée" d'une application ou d'un site web qui fonctionne en arrière-plan. Gère par exemple les données, la logique métier, et les interactions avec la base de données.

Image Raster : format d'image constitué d'une grille de pixels. Par exemple, le format PNG et un format d'image Raster.

Image Vectorielle : format d'image constitué de formes géométriques, assurant ainsi un zoom infini sans perte de qualité standard de web

Javascript : Langage de programmation de script standard du web moderne.

Framework : C'est un ensemble structuré de composants logiciels qui fournit une base pour développer des applications ou des logiciels. Il offre des outils, des bibliothèques, et des conventions qui facilitent le développement en réduisant la quantité de code à écrire et en assurant une certaine cohérence dans la structure du projet

Angular : Framework aidant à la création d'applications web dynamiques, ne nécessitant pas de recharge de page pour afficher de nouveaux composants. Son langage de programmation principal est le TypeScript, qui est un dérivé de Javascript.

Moteur cartographique : Application, logiciel ou librairie permettant l'affichage d'une carte dynamique.

CesiumJS : Moteur cartographique sous la forme d'une Librairie JavaScript, spécialisé dans l'affichage de cartes en 3 dimensions.

Maplibre GL JS : Moteur cartographique sous la forme d'une Librairie Javascript, très performant dans l'affichage de cartes en 2 dimensions.

Programmation Réactive : Méthode de programmation, où les éléments réagissent aux changements d'autres éléments, au lieu que ce soit les autres éléments qui appliquent eux mêmes des changements aux éléments.

5. Introduction :

Geomatys est une PME spécialisée dans le développement informatique, liée à la géospatiale et à la cartographie, développant à la fois des applications, les utilisant (tel que Aqualit, dont je parlerai en détail), mais aussi des outils facilitant leur exploitation.

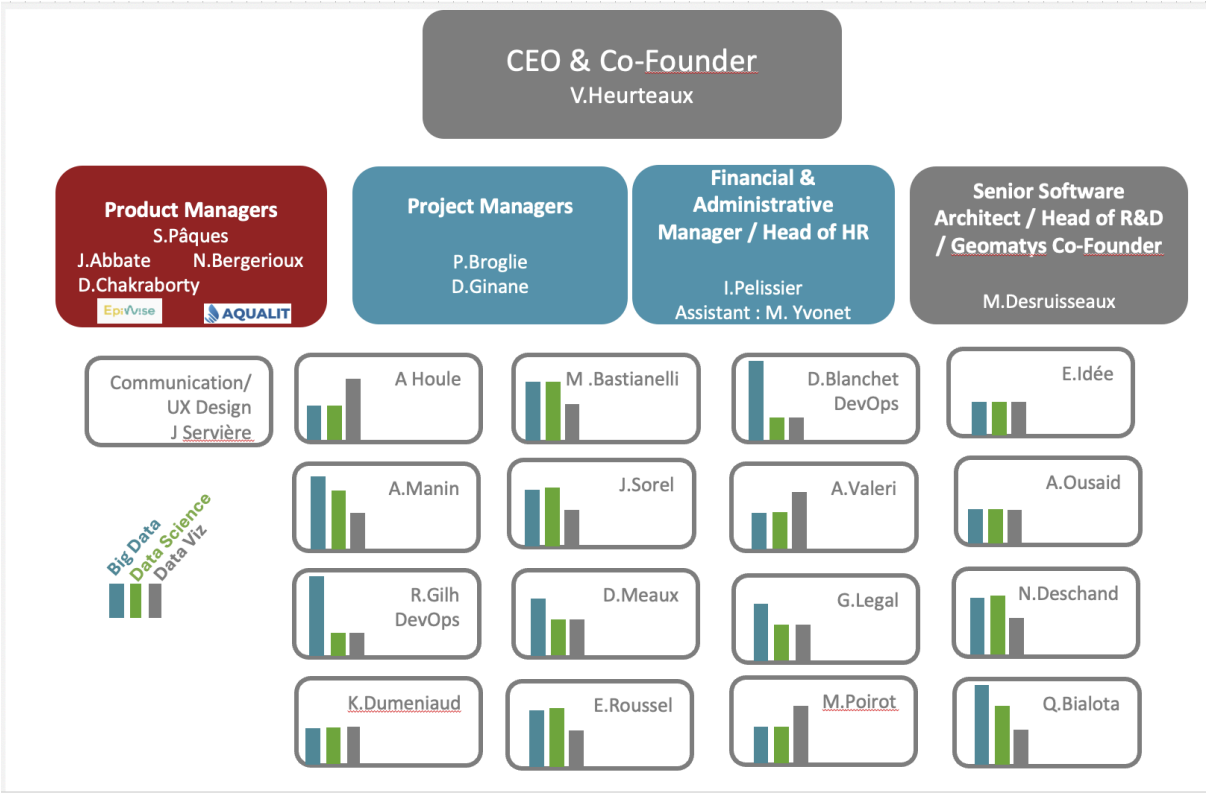


Fig.1 : Tableau du personnel de l'entreprise

Très certainement la réalisation la plus importante de Geomatys est la suite Examind.
Par exemple, Examind Server, qui possède aussi une version Open-Source nommée Examind Community, permet le traitement et le formatage de données géospatiales, permettant leur exploitation de manière optimale.

C'est donc dans le but de répondre à ces besoins que la plateforme Aqualit a été créée, et dont le développement et la maintenance a été confiée à Geomatys.

5.1 Explication de la mission du stage :

La production et la distribution d'eau font partie de ce groupe restreint de services qui se doivent de fonctionner aux plus hauts standards de qualité.
Afin de satisfaire ces standards, une surveillance rigoureuse de la qualité de l'eau à différents points du réseau de production et de distribution est nécessaire. Le regroupement de ces informations, ainsi que leur partage entre les divers acteurs devient alors un challenge à beaucoup de niveaux.

C'est dans le but de répondre à ces besoins que Geomatys développe et maintient la plateforme Aqualit, qui est une Application Web regroupant les données d'un grand nombre de capteurs liés à la qualité de l'eau, pour différents acteurs tels que SERPN, Chartres Agglomération, et le Conseil départemental de l'Eure. (Voir description détaillée en annexe)

Fig.2 : Présentation d'Aqualit

La navigation des données se fait au travers de plusieurs interfaces, parmi lesquelles une carte intéractive, affichant divers capteurs, que l'on peut sélectionner soit directement depuis la carte soit depuis un menu latéral.

C'est exclusivement sur cette partie cartographique de l'application que portait ma mission, son but étant de remplacer le **moteur cartographique** par un autre : Comprenez utiliser la librairie Maplibre GL JS plutôt que CesiumJS.
Bien que ce remplacement ne soit pas nécessaire au bon fonctionnement de l'application, il est souhaitable pour 2 raisons :

1. Raison de performances : Cesium est avant tout conçu pour de la 3D, et ne supporte pas les images vectorielles, ce qui en fait un choix sous-optimal pour l'affichage d'une carte en 2D

2. Raison d’esthétique : Toute la carte est affichée à partir d’images, ce qui à tendance à fortement se ressentir au travers du texte et des formes géométriques constituant la carte, visiblement pixelisées.

5.2 Analyse technique :

Afin d'améliorer ma compréhension des outils que j'allais utiliser, un accès à un projet utilisant Maplibre (Epiwise) m'a été accordé, en plus évidemment d'un accès à la repository git de Aqualit.

Ces deux projets sont des sites web dynamiques, utilisant le framework Angular, qui permet de développer en utilisant le langage TypeScript (JavaScript modifié qui possède un compilateur et un support de types), HTML et CSS/SCSS, et d'exporter le projet sous la forme d'un fichier JavaScript, qui sera lui interprété par le navigateur.

Afin d'assurer la bonne continuation du projet, j'ai suivi la convention de nommage et d'arborescence d'Angular Version 11 :

```
app/
├── shared/
│   ├── shared.module.ts
│   ├── shared.component.ts
│   ├── shared.component.html
│   └── ...
├── core/
│   ├── core.module.ts
│   ├── core.service.ts
│   └── ...
├── features/
│   ├── feature1/
│   │   ├── feature1.module.ts
│   │   ├── feature1.component.ts
│   │   ├── feature1.component.html
│   │   └── ...
│   └── feature2/
│     ├── feature2.module.ts
│     ├── feature2.component.ts
│     ├── feature2.component.html
│     └── ...
├── services/
│   ├── data.service.ts
│   ├── api.service.ts
│   └── ...
├── app.module.ts
├── app.component.ts
├── app.component.html
├── app.component.css
└── ...
```

Le fonctionnement du Framework Angular, notamment avec la Programmation Réactive rends difficile la représentation de l'application à l'aide des diagrammes appris durant ma formation. J'ai tout de même réalisé un diagramme de séquence représentant les interactions entre les classes MapComponent, MapService, et SensorMarkerService de la nouvelle version de l'application :

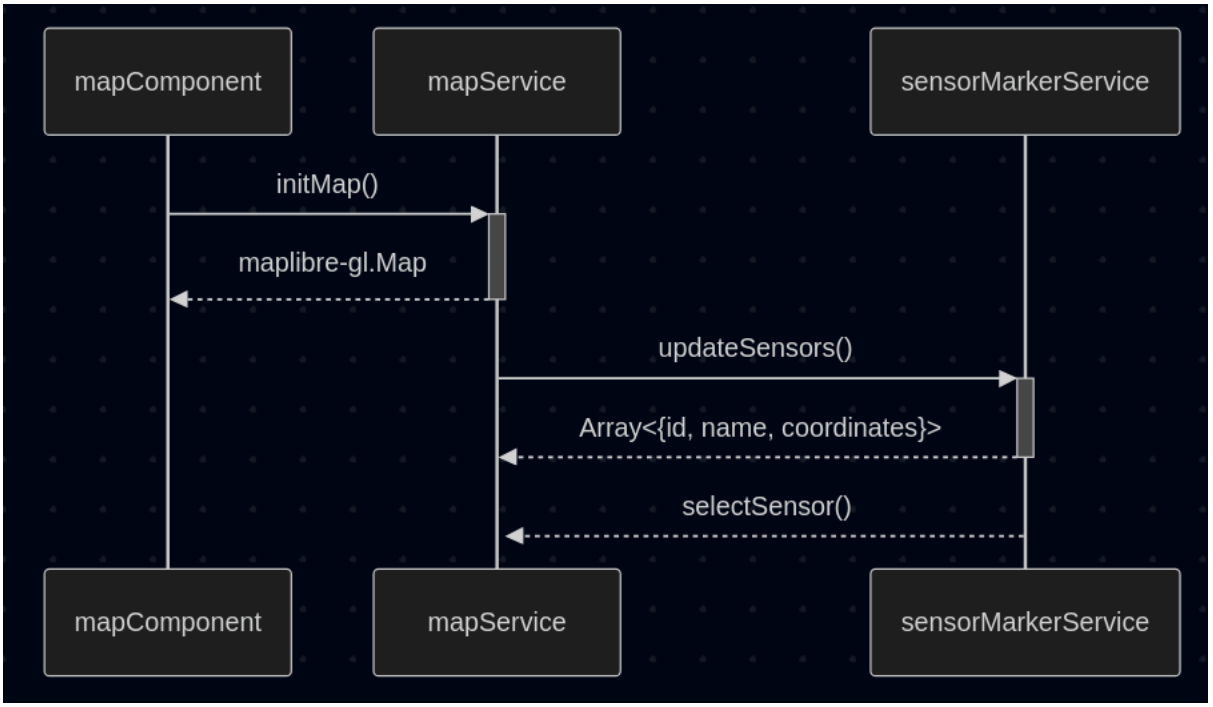


Diagramme de séquence du fonctionnement d'une partie de l'application

Explication dans le sens de lecture (gauche vers droite, haut vers bas) :

mapComponent contient l'élément HTML qui affiche la carte dans l'application web. La carte est initialisée à l'aide de la fonction publique **initMap()** de **mapService**, et renvoyé à **mapComponent**.

mapService est une classe dans Angular, qui gère tout ce qui est en rapport avec l'affichage de la carte, c'est-à-dire le contenu et le style de toutes les données de la carte.

sensorMarkerService est aussi une classe, qui gère l'initialisation ainsi que les changements des capteurs, en envoyant à **mapSevice** les données nécessaires pour les afficher.

5.3 Cahier des charges :

Le cahier des charges suivant a été rédigé au même moment que ce rapport, une estimation claire du temps que je passerai sur le projet n'étant pas importante, je me contentai simplement de réécrire l'existant en utilisant le nouveau moteur cartographique, et en gardant les fonctionnalités présentes.

Besoins fonctionnels :

- Changer de moteur cartographique
- Garder toutes les anciennes fonctionnalitées

Contraintes :

- Utilisation du Framework Angular (version 11)
- Utilisation de la librairie de cartographie Maplibre GL JS

Résultats attendus :

- Amélioration des performances
    - moins de charge côté client
- Améliorations de l'esthétique
    - utilisation d'images vectorielles et de texte => pas de pixelisation

6. Rapport technique :

Ma mission pour ce stage de 8 semaines était la suivante :

1. Me familiariser avec le fonctionnement d'une application web complexe développée avec le framework Angular, implémentant une carte dynamique à l'aide de la librairie CesiumJS.
2. Refaire la carte intéractive principale en utilisant la librairie Maplibre GL JS, tout en conservant les fonctionnalités précédentes.

6.1 Je ne connais pas Angular

La toute première difficulté et certainement la plus importante, était que je ne connaissais Angular que de nom. J'allais donc devoir apprendre à utiliser non seulement ce framework, mais aussi Typescript, qui est le langage utilisé pour la partie script d'Angular.

Heureusement, j'étais déjà relativement familier avec Javascript, dont Typescript n'est qu'une version modifiée, et l'on m'a fourni de très bonnes ressources pour apprendre Angular (Les tutos de angular.dev [7] et Angular Ninja [8]).

Cette partie formation a duré toute la première semaine, et je n'ai pas rencontré de problèmes techniques notables durant cette période.

6.2 Lancement de l'application

Le lundi de la deuxième semaine, je pensais avoir assez compris Angular pour commencer à travailler sur le projet. Nouveau problème : Je n'avais pas de compte sur le GitLab privé de l'entreprise, et personne de présent ce jour-ci ne pouvait m'en créer un. Afin que je me familiarise avec le code, j'ai reçu 2 fichiers zip provenant du dépôt git par clé USB (même contenu, mais sans historique).

Il n'y avait pas de réelle documentation, car le projet n'était pas si complexe que ça lorsque l'on est familier avec le code. J'ai passé quelques jours à comprendre comment fonctionnait l'application, et ce qui m'a de loin pris le plus de temps était tout ce qui est lié à de la cartographie. Étant donné que le projet fonctionnait dans un écosystème de plusieurs applications (à élaborer), j'ai aussi mis beaucoup de temps

à comprendre d'où venait certaines informations, comme la localisation de chaque capteur.

J'avais identifié dans le code les 2 éléments qui m'intéressaient le plus :

- La carte et son instanciation
- Le systèmes d'icônes customisées affichées sur la carte

Une fois ces éléments identifiés, j'ai trouvé leur équivalent, ce que ces équivalents avaient de commun entre eux, et ce qu'ils avaient de différent. :

Viewer de CesiumJS et **Map de Maplibre GL JS** :

Différences :

- le Viewer de CesiumJS utilise des données de type WMS et WMTS, ce qui veut dire que l'arrière plan de la carte est constitué d'images Raster [1], tandis que la Map de Maplibre GL JS peut utiliser des sources vectorielles (dans le même principe que les images SVG).

Points communs :

- Les deux possèdent des méthodes dont le fonctionnement est assez similaire, par exemple, elles ont toutes deux une méthode pour zoomer de manière fluide sur une coordonnée, et une méthode pour désigner une zone rectangulaire à partir de 2 longitudes et 2 latitudes, et d'ajuster la position et le niveau de zoom pour que la hauteur ou la largeur prennent l'entièreté de l'écran.

Billboard de **CesiumJS** et **Marker** de **Maplibre GL JS** :

Différences :

- Une Billboard peut avoir un texte qui s'affiche au survol, et bien que ce ne soit pas une fonctionnalité native des Markers, il est possible de recréer cette fonctionnalité en combinant d'autres éléments.

- Une Billboard ne peut être représenté que par une image, tandis qu'un Marker peut être représenté par un élément HTML, supportant tout type d'image, des styles avancées, ainsi que la possibilité d'y ajouter des EventListeners.

Points communs :

- Ils utilisent le même système de coordonnées (Longitude-Latitude).

- Tous leurs attributs sont dynamiques.

- Intéractibilité depuis la carte par des événements comme des cliques ou des survols.

6.4 Recréation de l'existant :

Après avoir eu accès au projet, cela m'a pris environ une semaine et demi pour refaire la carte et les icônes des capteurs affichés dessus. Cependant la carte semblait vraiment très lente, et surchargeait beaucoup trop le navigateur. J'avais attribué ces problèmes de performances au fait que je faisais tourner tous les modules de l'infrastructure sur ma machine, cependant, les performances s'amélioraient de manière exponentielle lorsque j'enlevai des capteurs.
La différence entre l'utilisateur qui avait ~250 capteurs et celui qui en avait ~25 était **très** importante.

Après quelques recherches en ligne [2], j'ai remarqué que je n'étais pas le seul à avoir des problèmes de performances lorsque je voulais afficher un grand nombre de Markers, et la réponse donnée était à chaque fois d'utiliser des Symbols.

Pour expliquer les symbols, je vais rentrer un peu plus dans les détails du fonctionnement de Maplibre :
Maplibre fonctionne avec des **Sources** et des **Layers** :

- Une Source contient des données dans un certain format. Les formats les plus utilisés sont ceux de type WMS, GeoJson, et Vector
- Un Layer est un calque qui affiche les éléments d'une Source selon des filtres, avec possibilité de leur appliquer des styles en fonction des attributs des éléments de la source.

Les Sources et Layers sont contenus dans le style de la carte, et possèdent chacun un identifiant unique. Lorsque j'ai instancié ma carte, j'ai renseigné un lien vers une liste de sources et de style au format, qui constituent l'arrière-plan de la carte. Par exemple, j'ai ici (exemple réel [4])un Layer qui affiche les éléments de la source "parks", ayant la classe "public_park", en leur donnant une couleur de remplissage vert clair, d'une opacité de 0.8, 1 et 0 signifiant visible et invisible.

```
{
  "id" : "landcover-grass-park",
  "type" : "fill",
  "metadata" : { "mapbox : group" : "1444849388993.3071" },
  "source" : "openmaptiles",
  "source-layer" : "park",
  "filter" : [ "==", "class", "public_park" ],
  "paint" : {
    "fill-color" : "#d8e8c8",
    "fill-opacity" : 0.8
```

```
  }
}
```

Cependant, (ce sera important pour plus tard), les Sources et Layers de l'arrière-plan ne sont pas séparés des Sources et Layers que j'ajoute moi-même. Mes Layers sont simplement ajoutés dans le style par dessus ceux existant, et l'ordre de mes sources n'est pas important.

Une fois que l'on a compris comment fonctionnent les Sources, les Layers, et le format GeoJson utilisé par les Sources, ajouter mes capteurs n'est pas très compliqué. Je dois simplement :

- Créer une source au format GeoJson, avec soit une liste vide, soit une liste de "Features" au format GeoJson, contenant :
  - Un **identifiant** ("sensors")
  - Des **propriétés** (variables associé à la Feature) : l'id du capteur et l'id de l'image qu'il doit utiliser
  - Une **géométrie**, dans mon cas, simplement une Longitude et une Latitude
- Créer un Layer avec :
  - Un **identifiant** ("sensors-layer")
  - un **type** ("symbol")
  - une **Source**, d'où proviennent les éléments à afficher ("sensors")
  - un **layout** décrivant des propriétés d'affichages avancées, tel que la propriété associée à l'image du capteur.

Clarification pour l'icône des capteurs : Contrairement à un Marker, dans lequel je peux simplement préciser le lien de l'image à utiliser, ici je dois en premier ajouter l'image à la carte. Pour celà j'utilise une méthode qui convertit le lien vers mon image en un objet, et à l'aide d'une seconde méthode de ma Map, associer un identifiant unique à cet objet. Je peux maintenant utiliser cette image dans mon Layer.

6.5 Sélection des capteurs :

Mes capteurs étaient maintenant affichés avec chacun leur image en fonction de leur type, et lorsque je cliquais sur l'un des capteurs dans le menu de gauche, cela entraînait un zoom de la carte sur la position de ce dernier.

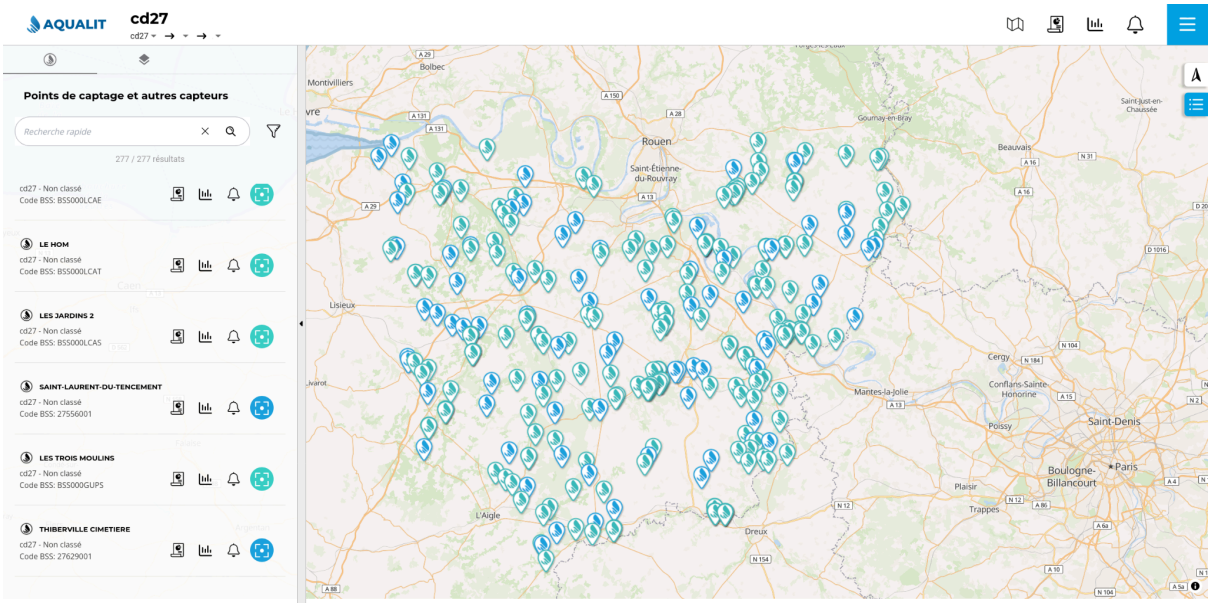


Fig.3 : Version améliorée de la Carte

Cependant, mes capteurs ne changeaient pas d'images lorsqu'ils étaient selectionnés. Avec les Marker, celà était relativement simple : j'avais créé une Map<K, V>, associant l'id d'un capteur à son objet Marker, et je n'avais qu'à modifier l'attribut css "background-image" de l'élément HTML lorsqu'il était sélectionné.

Ce n'était pas aussi simple pour les Symbols, le seul moyen était de modifier le GeoJson de ma Source, et ne changer que la propriété "imageId" de la feature possédant le même identifiant que le capteur sélectionné. Heureusement, Maplibre possède une méthode optimisé pour ce cas d'usage.

Je n'ai donc qu'à renseigner la source que je souhaite modifier, créer un objet de type *GeoJSONFeatureDiff* contenant :

- L'id de la Feature à modifier
- une liste (ne contenant qu'un seul élément dans mon cas) d'attributs et de leur nouvelles valeur ("imageId" et l'id de la version sélectionnée de l'image du capteur)

Et d'appliquer ce changement à ma Source, à l'aide de la méthode updateData().

```
const featureDiff: GeoJSONFeatureDiff = {
  id: this._selectedSensor?.id,
  addOrUpdateProperties: [
    {key: 'imageId', value: this._selectedSensor?.type}
  ]
};

const sourceDiff: GeoJSONSourceDiff = {
```

```
  update: [featureDiff]
};

this.mapService.map.getSource('sensors').updateData(sourceDiff);
```

(Le code ci-dessus est un extrait légèrement simplifié, par exemple, il ne modifie pas le capteur qui était précédemment sélectionné pour qu'il retourne à son icône non-sélectionné)

Pour ce qui est d'interagir avec les symbols au travers d'un clic de souris ou d'un survol, c'est relativement simple en utilisant un EventHandler sur la Map :

```
map.on('click', 'sensors-layer',  (location) => {
  const sensors = map.queryRenderedFeatures(location.point, {
    layers: ['sensors-layer']
  });
  . . .
  . . .
});
```

(**sensors** contient la liste des Features, chacune représentant un capteur, aux point précis où le clique a eu lieu. **sensors[0]** donne la feature la plus proche de clic)

Je peux ensuite récupérer l'id du capteur, et récupérer toutes les informations dont j'ai besoin avec la Map<K, V> associant un identifiant de capteur, avec des informations sur lui.
C'est ce que j'ai utilisé pour afficher un "Popup" qui affiche le nom et le type du capteur au survol de ce dernier, ou simplement pour le sélectionner depuis la carte.

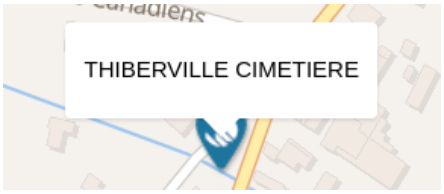


Fig.4 : Capteur sélectionné avec Popup

6.6 Clustering :

J'ai maintenant tous mes capteurs affichés, qui agissent comme je le souhaite, c'est-à-dire de la même manière que dans l'ancienne version de l'application, et surtout, sans aucun problème visible de performances.

Cependant, la personne qui a en quelque sorte le rôle de Product Manager m'a dit que ce serait bien d'avoir du "clustering" afin que ce ne soit pas aussi inutilement chargé visuellement.

Clustering signifie simplement regrouper plusieurs points en un seul, lorsqu'ils sont proches les uns des autres. Lorsque l'on zoom, les points s'éloigneront les uns des autres, et le cluster disparaitra.

Heureusement pour moi, Maplibre possède nativement cette fonctionnalité, il suffit simplement d'activer le clustering dans la déclaration de ma Source :

```
map.addSource('sensors', { // sensor source
  type: 'geojson',
      . . .
      . . .
  cluster: true,
  clusterRadius: 15,
});
```

De créer un Layer pour afficher et styliser le cluster :

```
map.addLayer({ // cluster layer
  id: 'clusters',
  type: 'circle',
  source: 'sensors',
  filter: ['has', 'point_count'], // only clusters are in the layer
  paint: {
    'circle-color': '#51bbd6',
    'circle-radius': 20,
  }
});
```

Un autre pour afficher et styliser le nombre d'éléments dans le cluster

```
map.addLayer({
  id: 'cluster-count',
  type: 'symbol',
  source: 'sensors',
  filter: ['has', 'point_count'], // only clusters are in the layer
  layout: {
    'text-field': '{point_count_abbreviated}',
    'text-size': 12,
  }
});
```

La page portant sur le clustering avec exemples [3] de la documentation de Maplibre m'a été plus que très utile pour réaliser cette fonctionnalité.

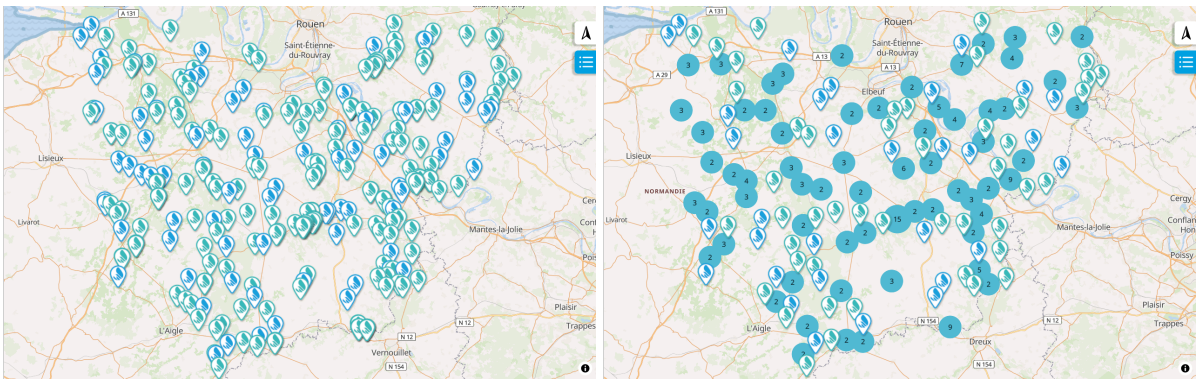


Fig.5 : Comparaison avec et sans clustering

6.7 Le capteur sélectionné qui disparaît avec le zoom :

Un problème qui est arrivé avec les clusters, était que maintenant, le capteur sélectionné disparaissait lui aussi dans un cluster si jamais je reculais trop.
Il fallait donc que je trouve un moyen d'empêcher le capteur sélectionné d'être clusterisé.
J'avais en premier pensé à afficher le capteur sélectionné dans un autre Layer, qui ne serait pas clusterisé, mais je me suis vite rendu compte que ce n'était pas possible, car un cluster s'applique à toute une source, et il n'est pas possible d'empêcher la clusterisation d'un Layer en particulier.
La solution était donc de créer une nouvelle source, qui ne contiendrait qu'un seul élément : le capteur sélectionné.

Lorsque le Subject du capteur sélectionné envoie une nouvelle valeur, 3 cas sont possibles :

1. Le capteur renvoyé est ***undefined*** (aucun capteur selectionné) et la Source **existe** :
    - suppression de la Source
    - suppression du Layer (dans cet ordre précis, car Maplibre refuse de supprimer une source qui est utilisé dans un Layer)

2. Le capteur renvoyé n'est **pas *undefined***, et la Source **existe** :
    - modification de la source, de la même manière qu'expliqué précédemment, en utilisant la méthode updateData()

3. Le capteur renvoyé n'est **pas *undefined***, et la Source **n'existe pas**
    - création de la Source
    - création du Layer

Il fallait aussi afficher le "Popup" lorsque l'on passait la souris au-dessus du capteur. Cela fonctionne car l'EventListener détectant le survol du capteur du dessous fonctionne toujours, mais seulement tant que ce capteur n'est pas clusterisé.

J'ai donc apporté les modifications suivantes :

- Ajout d'une condition dans l'event listener pour d'affichage du popup, afin qu'il ne fasse rien si le capteur survolé est le capteur sélectionné
- Création d'un event listener sur la version sélectionné du capteur, afin qu'il affiche le popup au survol

Étant donné que la version sélectionnée du capteur serait de toute façon affichée par dessus, j'ai supprimé la partie du code qui changeait la source "sensors" (changeait la version non sélectionnée de l'image pour une version sélectionnée).

Dézoom sur les capteurs :
Une feature de la carte précédente que j'ai initialement eu du mal à refaire est le dézoom lorsque je désélectionne un capteur. Ce zoom était fait de manière à être au centre de tous les capteurs, et les incluait tous dans l'écran.
J'avais initialement fait un zoom sur la longitude et la latitude moyenne de tous les capteurs, ce qui le plaçait au centre, mais ne garantissait pas qu'ils soient tous inclus dans l'écran.

J'ai donc demandé de l'aide sur ce point, et l'on m'a redirigé vers une objet standardisé en cartographie web : la Bounding Box (ou Bbox pour faire court).

Le format Bbox de Maplibre possédait 2 coordonnées : une longitude/latitude sud-ouest, et une longitude/latitude nord-est.
Ces deux points forment un rectangle, sur lequel il est possible de zoomer en lui faisant prendre tout l'écran.

Afin de calculer cette Bbox, dans la boucle for que j'utilisais pour ajouter les capteurs au GeoJson, j'ai ajouté 4 conditions, afin de trouver :

- La Latitude (axe nord-sud)
  - la plus au nord
  - la plus au sud
- La Longitude (axe ouest-est)
  - la plus à l'ouest
  - la plus à l'est

```
if (l.coordinates[0] > this.sensorsBbox._ne.lng)
    this.sensorsBbox._ne.lng = l.coordinates[0];
if (l.coordinates[1] > this.sensorsBbox._ne.lat)
    this.sensorsBbox._ne.lat = l.coordinates[1];
if (l.coordinates[0] < this.sensorsBbox._sw.lng)
  this.sensorsBbox._sw.lng = l.coordinates[0];
  if (l.coordinates[1] < this.sensorsBbox._sw.lat)
```

```
    this.sensorsBbox._sw.lat = l.coordinates[1];
```

La Bbox était donc un array de type [[number, number], [number, number]], avec les valeurs suivantes :
[
    [Latitude la plus au sud, Longitude la plus à l'ouest],
    [Latitude la plus au nord, Longitude la plus à l'est]
]

J'ai cependant remarqué que lorsque j'effectuais un zoom sur cette Bbox, tous les capteurs étaient bien inclus, mais ceux sur la bordure étaient en quelque sorte coupés en deux, car la bordure de la bbox était au centre de l'icône.
Afin de régler ce problème, j'ai créé une fonction qui élargit la Bbox par un facteur donné (exemple 1.2), afin de l'élargir suffisamment pour que chaque capteur soit visible en entier.

```
private bboxEnlarge(factor: number) {

  const maxLng = this.sensorsBbox._ne.lng;
  const maxLat = this.sensorsBbox._ne.lat;
  const minLng = this.sensorsBbox._sw.lng;
  const minLat = this.sensorsBbox._sw.lat;

  const centerLng = (minLng + maxLng) / 2;
  const centerLat = (minLat + maxLat) / 2;

  const width = maxLng - minLng;
  const height = maxLat - minLat;

  const lrgWidth = width * (factor);
  const lrgHeight = height * (factor);

  this.sensorsBbox._ne.lng = centerLng + (width / 2) + (lrgWidth / 2);
  this.sensorsBbox._ne.lat = centerLat + (height / 2) + (lrgHeight / 2);
  this.sensorsBbox._sw.lng = centerLng - (width / 2) - (lrgWidth / 2);
  this.sensorsBbox._sw.lat = centerLat - (height / 2) - (lrgHeight / 2);
}
```

6.8 Les capteurs qui se chevauchent :

Le Product Manager (et moi-même) avait remarqué un problème qui était là depuis la version précédente de l'application : Il arrive que des capteurs soient exactement

aux mêmes coordonnées. Nous l'avions remarqué grâce aux clusters, car certains indiquaient contenir 2 éléments, mais lorsque l'on effectuait un zoom suffisant pour que le cluster ne s'affiche plus (chose faite automatiquement quand on clique sur un cluster), il ne semblait n'y en avoir qu'un seul.

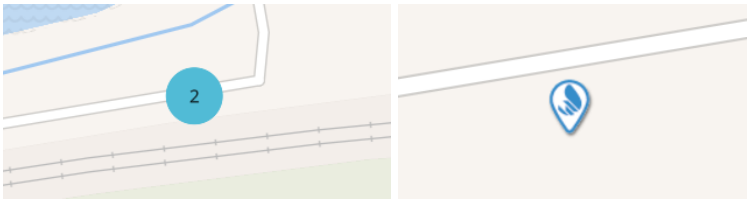


Fig.6 : 2 capteurs l'un au dessus de l'autre

Le Product Manager m'a donc demandé si possible de faire en sorte que les capteurs qui se chevauchent soient espacés lorsque l'on passe la souris par dessus.

Après quelques recherches, je n'ai pas trouvé de fonctionnalités natives à maplibre pour ce cas exact d'espacer des Symbols qui se chevauchent.
Je suis donc parti pour autre chose : modifier l'EventListener de clic sur un capteur, afin que :

- Si tous les capteurs à l'endroit précis du clic sont aux mêmes coordonnées :
  - La géométrie (coordonnée) des Symbols est modifié de manière à ce qu'ils soient affichés espacés, de manière centré autour de la géométrie originale
- Sinon :
  - Le capteur est sélectionné ou désélectioné

Au final la consigne n'a pas *exactement* été respectée, mais l'opération de changer la géométrie + l'actualisation de la carte est trop longue pour pouvoir être quelque chose comme les informations d'un capteur qui apparaissent et disparaissent au survol de souris.

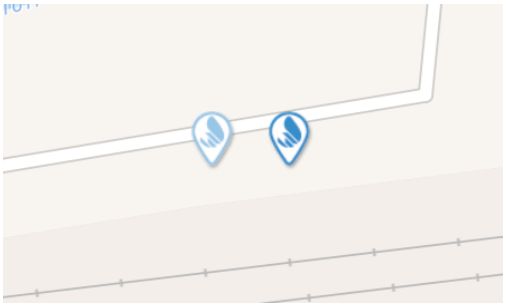


Fig.7 : Capteurs espacés

6.9 Filtrer les layers :

Ne pas avoir de cahier des charges à ses avantages, par exemple, je n'ai pas à me soucier de quelconque deadline, je ne me sens pas submergé par une grosse quantité de travail quand je commence.

Cependant, cela a aussi des inconvénients, comme je l'ai découvert ici : Je pense avoir pratiquement fini le projet, jusqu'à ce que je découvre une nouvelle fonctionnalité de l'application, que je dois aussi refaire.

Dans le menu de gauche, il est possible de filtrer les capteurs selon plusieurs critères

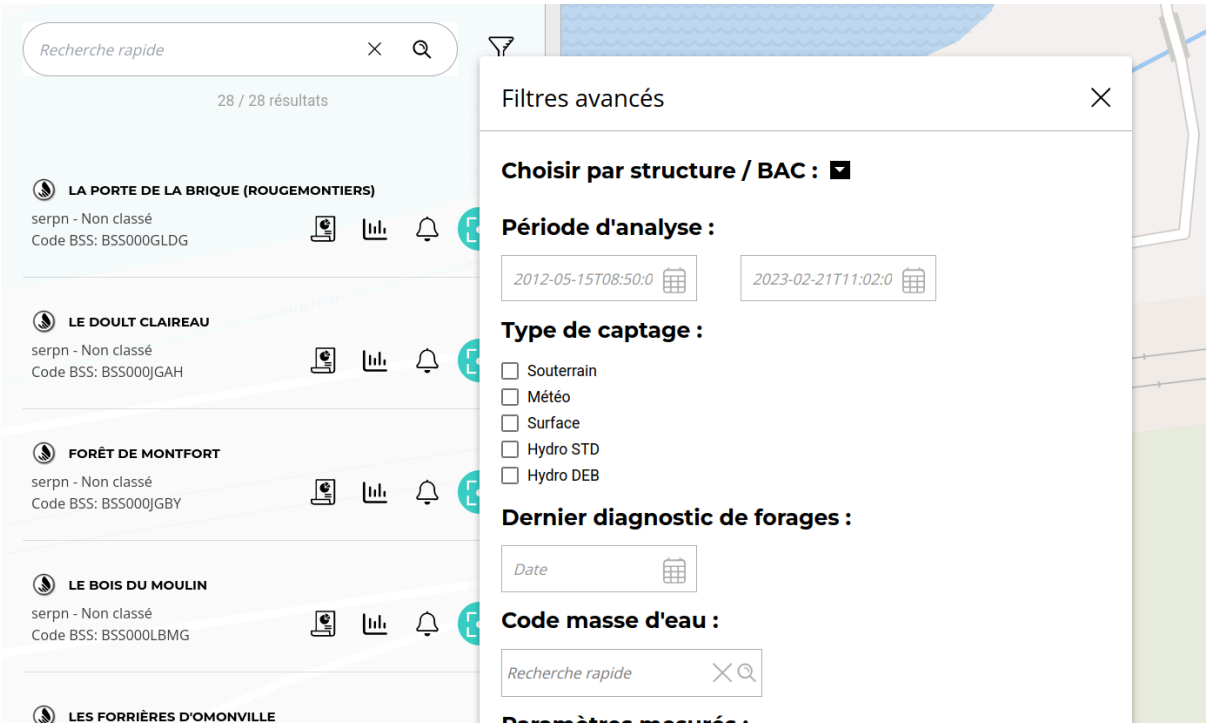


Fig.8 : Menu de filtrage

Ce que je ne savais pas, c'est qu'une fois le filtre appliqué, seuls les capteurs correspondants aux filtres doivent restés affichés sur la carte.

Heureusement, toutes les fonctions permettant ceci étaient déjà faites, et je n'ai pas eu besoin de faire beaucoup de modifications pour que celà fonctionne.

6.10 Layers WMS customs :

Ceci est un autre exemple de fonctionnalité que je découvre, mais celle-là était beaucoup plus conséquente que l'affichage des layers après filtrage.

J'avais en premier remarqué qu'il y avait une légende des données de la carte, et j'ai cru comprendre que c'était quelque chose d'associé à la carte CesiumJS, et qui n'avait apparemment pas d'équivalent dans MapLibre. Je me suis d'ailleurs dit que ça ne devait pas être important, car quelle utilité d'avoir une légende étant donné que la carte ne sert que de point de repérage sur la position des capteurs ?

Fig.9 : Une des légendes de la carte

J'ai donc demandé ce que je devais faire par rapport à cette légende, m'attendant à ce qu'on me dise d'oublier ça et que si ce n'est pas techniquement possible, ça ne posait pas de problème de l'enlever.

En réalité, la légende n'était pas la légende de la carte, mais la légende de calques que l'on peut ajouter à la carte. Et je ne savais pas qu'il était possible d'ajouter des calques customs depuis un menu.

Fig.9 : Menu d'ajout de calques customs

De plus, ces calques étaient sous le format WMS, qui sont représentés par des images formés par des pixels (comme le format png), et non par des vecteurs, comme l'était ma carte.

Maplibre supportait évidemment les calques WMS, donc je savais que c'était faisable, mais lorsque je remontais les sources de la provenance de ces calques WMS, le code que je voyais n'avait aucun sens, et je ne comprenais pas comment quoi que ce soit pouvait utiliser ces méthodes et fonctionner.

```
export declare class MapContextService<T> {
  protected _mapTree: MapLayers;
  private update;
  updated: Observable<MapContextUpdateEvent<T>>;
  get mapTree(): MapLayers;
  get mapContext(): Array<MapLayer<T>>;
  constructor();
  init(mapTree: MapLayers): void;
```

```
  addMapTreeItem(item: MapItem, position?: number): void;
  removeMapTreeItemAtPosition(position: number): void;
  moveMapTreeItem(startPosition: number, endPosition: number): void;
  . . .
  . . .
}
```

Le code n'était constitué que de méthodes vides, mais qui étaient utilisées par des éléments de mon application.

Je suis donc allé demander de l'aide, et ayant plus d'expérience que moi en TypeScript, le problème a vite été identifié : je regardais les fichiers compilés, qui ne contenaient que la signature des méthodes et le type des variables.
En fait, lorsque l'on installe des dépendances, leur forme compilée est placée dans un dossier non-versionné de mon projet, et c'est aux méthodes et variables de ces fichiers typescript compilés que mon IDE me renvoyait lorsque j'effectuais l'action *go to reference/definition*.

On m'a donc donné accès aux dépôts Gitlab de ces dépendances, afin que je comprenne comment tout fonctionne. Finalement ce n'était pas extrêmement compliqué, mais pas "facile" non plus.
Le menu de gauche émettait à l'aide d'un Subject un Array d'objet customisé "MapLayer" à chaque changement, comme un changement d'opacité, de visibilité, d'ajout/suppression d'un calque, ou bien un changement d'ordre des calques.

Cet array représentait les calques dans leur ordre, ainsi que tous leurs attributs.

Fig.10 : Array de MapLayer, affiché dans la console de Mozilla Firefox
(il n'y a qu'un seul calque dans cet exemple)

Et non pas une liste de changements comme le ferait Git. La chose la plus simple à faire aurait donc été de supprimer les calques, et de les ré-ajouter, de cette manière je n'aurais pas eu à chercher les différences et à les appliquer.
Pour des raisons assez évidentes (opérations inutilement coûteuses en ressource ainsi que visuellement), j'ai tout de même préféré créer une fonction smartUpdate(),

qui pour chaque Calque, déterminera le minimum d'opérations que j'ai besoin de faire. Par exemple :

- Un calque qui n'était pas là avant est présent :
    - Le calque est créé
- Un calque qui était là avant n'est plus présent :
    - Le calque est supprimé
- L'opacité d'un calque a changé :
    - je modifie son opacité

Pour l'ordre des calques, j'ai trouvé que la solution la plus simple était d'appeler à chaque smartUpdate, une fonction qui déplace chaque calque en précisant en dessous duquel ils doivent être positionnés.

6.11 Styles de la carte :

A ce moment là (je peux le dire maintenant que j'ai du recul), toutes les fonctionnalités de l'ancienne version, étaient présente dans mon application. Mon travail consistait donc maintenant à changer à intégrer de nouvelles features demandés par le Product Manager, et à faire du travail d'optimisation.

J'avais vu lundi matin un message du Product Manager datant de vendredi soir, me demandant d'ajouter la possibilité de changer de style de la carte [4].

Je m'étais dit que ce serait très facile, j'avais déjà essayé plusieurs styles de carte, et je n'avais qu'à changer un attribut à l'instanciation de la carte. Après de courtes recherche (sous la forme de taper **this.map.*style*** et de laisser mon IDE me montrer les possibilitées), j'ai même trouvé une méthode setStyle(*url: string*), j'allais surement avoir fini avant même que le Product Manager n'arrive le matin.

Vous rappelez-vous lorsque j'ai dit que l'arrière-plan de la carte n'était d'aucune manière séparé des autres Sources et Layers ? A ce moment là, je ne le savais pas, et il s'avère que par défaut, changer le style de la carte au travers de la méthode setStyle() revenait à supprimer **toutes** les Source ainsi que tous les Layers existant de la carte, et d'ajouter ceux provenant du nouveau style.

Heureusement, la méthode setStyle peut prendre un autre argument, qui est en lui-même une fonction : transformStyle [5].

Cette fonction prend deux arguments : *previousStyle*, qui est le style actuel de la carte, et *nextStyle*, qui est le nouveau style que nous souhaitons appliquer. La syntaxe est assez simple, une fois que l'on sait que :

- …myArray “largue” en quelque sorte tout le contenu de l’array. Par exemple :
  const sansUn = [2, 3, 4];
  const avecUn = [1, …sansUn];
  // avecUn aura comme contenu [1, 2, 3, 4]

- Chaque déclaration est soit un ajout, soit un override de ce qui a été déclaré avant.

Le but est donc de récupérer les Sources et Layers qui ne font pas partis du style original (soit les sources et layers relatifs aux capteurs, ainsi qu’aux calques WMS ajoutés)
Le code de cette fonction fait à peine 30 lignes, et la partie la plus complexe est celle qui transforme la liste de mes sources en un format accepté, ce n’est pas central au fonctionnement, je vais donc passer l’explication de cette partie.

Voici une version simplifié d’une partie du code :

```
this.map.setStyle(style, {
  transformStyle: (previousStyle, nextStyle) => {
    ...nextStyle,
    sources: sources,
    layers: [
              ...nextStyle.layers,
              ...previousStyle.layers.filter(layer =>
                customIds.layerIds.has(layer.id)
              )
            ]
  }
});
```

Explications :

- Le premier argument *style* est l’url vers le nouveau style de la carte
- Le deuxième argument contient transformStyle, qui est une fonction dans laquelle je peux préciser quels éléments de quel style garder ou modifier
    - Premièrement je déclare vouloir utiliser tous les éléments contenus dans nextStyle
    - Ensuite j’override les sources de nextStyle, pour une version des sources contenant les sources de nextStyle, ainsi que les sources liées aux capteurs et aux calques WMS (ici volontairement masqué pour plus de clarté)

- Les layers sont aussi override :
    - Je déclare utiliser tous les Layers de nextStyle
    - Je déclare ajouter les Layers de previousStyle, dont l'id se trouve dans la liste d'Id de layers que j'ai moi-même ajouté

Quelque chose de très bien avec la fonction setStyle, c'est qu'elle remplace de manière efficace les éléments (d'une manière similaire que ma fonction smartUpdate()), et cela se voit lorsque l'on passe de certains styles à un autre. Par exemple, on peut voir des axes routiers simplement changer de couleurs, au lieu d'être complètement rechargés.

6.12 Fenêtre d'informations :

Le Product Manager m'a aussi demandé d'afficher une petite fenêtre contenant des informations sur un capteur lorsqu'il est sélectionné, et pouvant plus tard être amélioré en y ajoutant des informations comme une image du capteur. La fenêtre devrait s'afficher lorsqu'un capteur est sélectionné, disparaître lorsque aucun capteur n'est sélectionné, pouvoir être fermé, et pouvoir être déplacé sur la carte, à la manière d'une fenêtre popup sur n'importe quel système d'exploitation moderne muni d'une interface graphique.

J'avais le candidat parfait pour ça : un Marker, car si vous vous souvenez, c'est un point sur la carte, dont les attributs sont dynamiques, et qui peut contenir des éléments html complexes avec leur propre style.

Après quelques heures de travail, j'ai réalisé cette fenêtre :

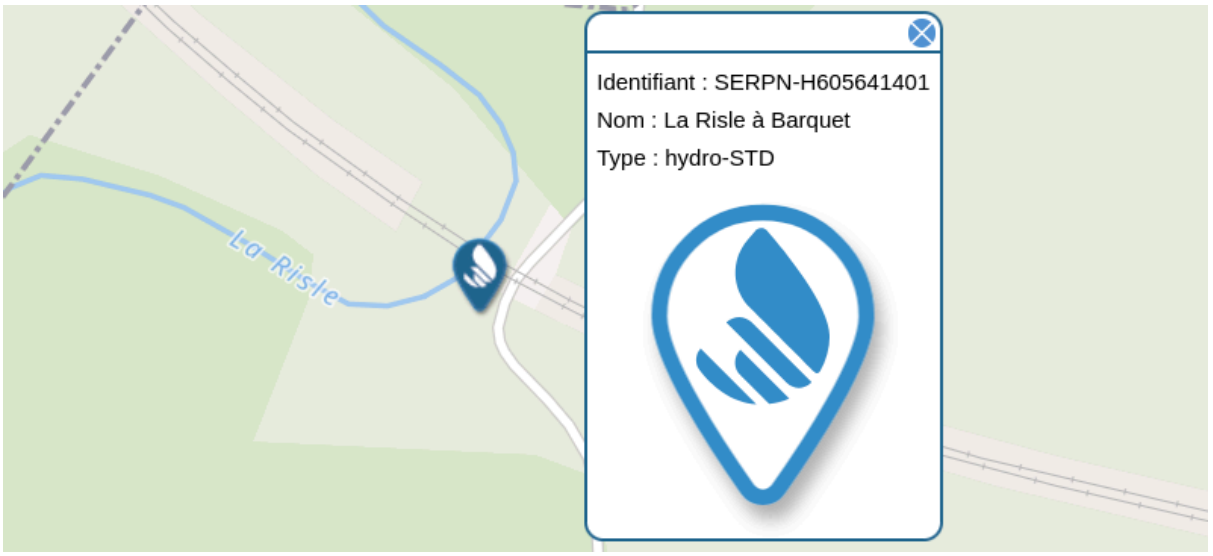


Fig.11 : Fenêtre d'information

Elle est affiché sur la droite d'un capteur lorsqu'il est sélectionné, en le plaçant aux coordonnées du capteur, avec un offset le décalant sur la gauche

```
this.infoWindow
.setLngLat(sensorInfos.coordinates)
.setOffset([150, 0])
.addTo(this.mapService.map);
```

Il n'est possible de la déplacer que en maintenant clic gauche sur la barre du haut (comme sur beaucoup de systèmes d'exploitations), ce qui permet donc de pouvoir sélectionner le texte des informations affichées, sans la déplacer.

```
// window is only movable when moving using the top bar
topBar?.addEventListener('mouseenter', () =>
  this.infoWindow.setDraggable(true)
);
topBar?.addEventListener('mouseleave', () =>
  this.infoWindow.setDraggable(false)
);
```

6.13 Importants problèmes de performance :

Cela faisait quelques temps que j'avais remarqué des problèmes de performances, principalement sous la forme de délai de mise à jour de la carte (délai d'apparition des nouveaux détails de la carte lorsque l'on zoom, et changement de l'icône du capteur lorsqu'il est sélectionné). Ils étaient relativement minimes lorsque j'étais connecté à un utilisateur ne possédant que 25 capteurs, mais devenaient très importants (délais allant jusqu'à plusieurs dizaines de secondes), lorsque j'étais sur un utilisateur ayant aux alentours de 250 capteurs.

La seule solution viable mais insatisfaisante que j'avais trouvé était d'utiliser un Marker (qui je le rappelle est un conteneur d'élément HTML ajouté par dessus la carte), contenant la version sélectionnée de l'icône d'un capteur, que j'ajouterai par dessus sa version non-sélectionné, affiché par le Symbol. De cette manière, le capteur paraissait être sélectionné de manière instantanée (ce qui était le cas en back-end), de cette manière, aucune modification chronophage de la carte n'aurait à être exécutée.

Cependant, il y avait un léger offset entre l'icône sélectionnée et l'icône non-sélectionnée, qui ne se résolvait que lorsque la carte était complètement

chargée (ce qui je le rappelle pouvait prendre plusieurs dizaines de secondes).

Fig.12 : Offset de l'icône sélectionné par rapport l'icône non-sélectionnée

Ajouté à cela que persistait le problème de la lenteur de chargement de la carte, et qu'il était évident que ce problème empirait de manière significative, plus j'avais de capteurs à afficher.

Étant évidemment un problème lié à Maplibre, j'ai cherché des solutions, et une arrivait en tête comme la solution miracle : Les Vector Layers. Je ne vais pas trop m'attarder sur l'explication puisque ça n'a pas été la solution finale, mais pour faire simple, au lieu que ma source soit constitué d'objet dans du Json, elle contiendrait un lien vers un serveur, distribuant mes données sous la forme de fichier binaires compressés, permettant ainsi une vitesse et une efficacité de traitement optimal des données.

Mais durant le week end avant ma dernière semaine de stage, je suis retombé sur un tutoriel de la documentation Maplibre [3] alors que j'écrivais mon rapport de stage. Ce tutoriel donnait un exemple intéractif relativement similaire au mien, utilisant un grand nombre de données clusterisées, sous le format GeoJson, à la différence que cet exemple était extrêmement fluide et sans aucun délai notable. La seule différence que je pouvais voir avec mon application était que mes Symbols avaient chacun une image PNG, contrairement à l'exemple qui n'avait soit des chiffres soit des points vides.

Cela faisait parfaitement sens, étant donné que le problème semblait provenir du rendering, et non de la lenteur de modification des données. Tout ce que j'avais à faire, c'est donc créer une police d'écriture customiser, et d'utiliser les symboles créées avec la bonne couleur, au lieu d'une image.

J'ai donc testé lundi matin, en remplaçant les images par le type du capteur, mais les performances ne s'étaient pas du tout améliorées, et je savais donc maintenant qu'il devait y avoir une autre différence entre mon application, et celle de l'exemple.

Le plan était simple : supprimer une par une les fonctionnalités liées à la carte, jusqu'à ce que que les performances sembles normales, afin de trouver l'origine du problème.
Dans l'ordre j'ai :

- Supprimé toute la logique d'ajout de données la Source contenant les capteurs,
- Supprimé les Layers,
- Supprimé la Source,
- Supprimé la partie qui initialise et met à jour les données dans une autre partie de l'application.

Même avec une carte complètement vide, aussi bien en front-end qu'en back-end, le chargement de la carte restait plus lent d'un profil à l'autre. J'ai demandé de l'aide à des développeurs qui connaissaient mieux le projet que moi, mais ils ne comprenaient pas non plus comment cela était possible.

Après plusieurs heures d'investigation, j'ai finalement supprimé un élément qui rendait ma carte vide parfaitement fluide, sur n'importe quel utilisateur, et en quelques minutes j'ai trouvé le responsable exact de tous mes problèmes : Le menu de gauche.

3 des 4 éléments du menu de gauche causaient des problèmes, chacun pour les raisons suivantes :

La barre de recherche :

- Problème : Lorsque la barre de recherche est active, une liste des possibilitées de recherche s'affiche, sous la forme d'un menu déroulant, avec autant d'éléments qu'il y a de capteurs. Cependant, tous ces éléments (je testais cela avec ~250 capteurs) restaient chargés, même lorsqu'ils n'étaient pas affichés. Cette illusion est créée par le Material Design utilisé. Les Material Designs sont un ensemble d'éléments html complexes faisant partie d'Angular, facilitant grandement la création et la gestion de certains éléments html complexes, comme dans mon cas d'un auto-complete.

- Solution trouvée : modifier l'attribut Angular ***ngFor**, pour utiliser une liste vide comme source de création des éléments contenant les propositions (n'en affichant donc aucun), lorsque la barre de recherche n'est pas active.

```
<mat-option
*ngFor="let option of (isInputFocused ? filteredOptions$ :
emptyFilterOptions)
>
  {{ option.name }}
</mat-option>
```

Le menu de filtrage :

- Problème : comme précédemment, le menu de filtrage est un objet complexe, qui s'affiche lorsque je clique sur un bouton, mais comme les propositions de

recherche, l'élément reste chargé.

- Solution : utiliser l'attribut dynamique ***ngIf** d'Angular, qui ne crée l'élément html que si la condition (dynamique) est vraie. Dans mon cas, je vérifie si l'élément parent (de classe sensor-filters-menu) est affiché. Je n'ai pas fait ça pour la barre de recherche, car il n'est pas possible d'avoir un ***ngIf** et un ***ngFor** sur le même élément.

```
<mat-menu #sensorFilters class="sensor-filters-menu">
  <app-sensor-filters-menu *ngIf="isFilterMenuOpen()" . . . . . />
</mat-menu>
```

La liste de capteurs :

- Problème : Il y a trop d'éléments html chargés. Plusieurs icônes, boutons, balises p, avec plusieurs attributs dynamiques, multiplié par le nombre de capteurs, comme 250 dans mes tests, et cela se ressent au niveau des performances de l'application. Le problème était surtout que j'avais beaucoup d'éléments sur lesquels Angular écoutait automatiquement tout changement. Ces écoutes étaient coûteuses, et le seul changement qui pouvait arriver était la couleur du bouton qui change lorsqu'il était sélectionné.
- Solution : Par défaut, Angular détecte les changements d'un composant de manière automatique, qui compare pour trouver des changements à chaque clique, survol, requête, opération asynchrone … . Cela est souhaitable, mais pas partout, j'ai donc override la stratégie de détection de changement dans le composant **<app-sensor-filters-menu/>**, pour utiliser celle qui ne vérifie qu'à l'initialisation, et puis plus jamais. Pour le changement de couleur du bouton, je peux simplement modifier le css directement en utilisant du js/ts vanilla, selon si le capteur selectionné est celui de l'élément. Avec ce changement, sur 250 capteurs, je suis passé de 8 à 4 secondes entre la sélection d'un capteur et l'affichage de l'icône sélectionnée, et encore moins avec moins de capteurs.

```
@Component({
  selector: 'app-sensor-item',
  templateUrl: './sensor-item.component.html',
  styleUrls: ['./sensor-item.component.scss'],
  changeDetection: ChangeDetectionStrategy.OnPush,
})
```

7. Méthodologie Organisationnelle :

7.1 Organisation du travail dans l'entreprise :

Je n'ai pas eu de règles strictes concernant d'éventuelles deadlines, le projet se faisait à mon rythme, et le suivi n'était pas intrusif, se limitant à des visites du Product Manager afin de s'assurer que le projet avance bien (ce qui a toujours été le cas), et aux retours sur mes Merge Requests.

Moins officiellement, le développeur en charge de mes merge request avait une très bonne idée de l'avancée du projet étant donné que j'allais le voir assez souvent lorsque j'avais des problèmes que je n'arrivais pas à résoudre.

Étant donné que j'étais seul à travailler sur le projet, Il n'y avait aucun besoin d'utilisation d'outil de répartition des tâches.

En raison de mon manque de familiarité avec le projet (Aqualit) et les technologies utilisées (Angular, Typescript, MapLibre), il était impossible de planifier en accord avec des estimations de durée des tâches.

7.2 Versioning avec Git :

Puisque j'étais à la fois dans la phase "apprentissage", "expérimentation", et "production du code", j'ai préféré ne pas "polluer" le Git avec toutes mes expérimentations, et uniquement faire un commit + push lorsque j'aurais quelque chose d'assez propre, par exemple sans grosse partie de code théoriquement inutilisé mais sans laquelle le projet ne fonctionne plus du tout.

Je pense que c'était la bonne chose à faire au vu de tous les changements que j'ai fait, (avec le recul) du peu que je comprenais, et du fait que l'affichage du code modifié par rapport au précédent commit git de mon IDE m'aidait beaucoup à me repérer quand je n'étais pas très familier avec le projet.

C'est donc à partir de ce moment-là (je rappelle que je rédige ce rapport technique de manière linéaire) que j'ai fait le premier commit + push sur une nouvelle branche.

En prenant exemple sur les anciens commits, mes messages étaient structurés de la manière suivante :
"action(lieu): précisions", avec :

- action : l'action principal de ce commit, le plus souvent "feat", "fix" ou "refactor"

- lieu : où étaient les ajouts/changements, je mettais tout le temps “cartography” étant donné que c’est essentiellement ce sur quoi je travaillais.
- précisions : message décrivant les changements

exemple réel : *feat(cartography): selected sensors icons are now always displayed on top of any other icon, including clusters*

J’ai remarqué que tous les commentaires nom de variables/fonctions, commits, et retour de merge requests étaient écrits en anglais, j’ai donc fait de même.

8. Conclusion :

8.1 Bilan du travail réalisé :

En précisant qu'il me reste encore 2 jours de stage à l'heure où j'écris ceci, j'estime le travail fini à 98%, possiblement fini avant la fin de mon stage. Les objectifs initiaux d'amélioration des performances et de l'esthétique de la carte ont parfaitement été atteints :

- La carte est plus fluide, ses mises à jour (nouvelles données qui s'affichent/disparaissent selon le niveau de zoom) sont plus rapides, le nombre de capteur n'ayant qu'un effet minime sur les performances.

- La carte est plus esthétique, aucune pixellisation visible grâce aux images vectorielles de Maplibre, et l'amélioration des performances a grandement amélioré l'expérience utilisateur.

J'ai de plus implémenté de nouvelles fonctionnalités sur demande du Product Manager, tels que le clustering de capteurs, le changement de style de la carte, et la fenêtre d'information.

8.2 Bilan des apprentissages :

Ce stage m'a énormément appris en termes de technologies et de bonnes pratiques dans le domaine de la réalisation d'applications web, et de leur déploiement, et sur une autre note dans le domaine de la cartographie.

Plus précisément :

- Utilisation avancée de **Docker** :
    - Compréhension exacte des principes et de l'utilisation des Dockerfile, images, conteneurs, et d'orchestration via docker compose
    - Orchestration de plusieurs conteneurs communicants ensemble

- Utilisation de **Git** dans un contexte de collaboration en entreprise :
    - Bonnes pratique de nommage
    - Bonnes pratique de procédure (création de branche pour chaque feature, développement sur plusieurs branches en parallèle, résolution de conflits)
    - Procédure de Merge Request pour l'intégration du code.

- Apprentissage et utilisation avancée du Framework Angular, et par extension, le langage TypeScript :
    - Utilisation de principes de programmation réactive, principalement avec la librairie RXJS (Reactive Extensions for JavaScript)
    - Gain de pratique avec le langage Typescript, se transcrivant très bien

en compétences en JavaScript
    - Programmation en asynchrone
- Apprentissage et utilisation du SCSS (Sassy CSS)
- Apprentissage et utilisation de technologies et principes liées à la cartographie en web :
    - Formats de donnée : GeoJson, WMS, fichiers PBF
    - Système de coordonnées : Latitude et Longitude (radians et degrés)
    - Objets cartographiques : Bounding Box

8.3 Signatures :

Signature du tuteur de Stage :

Vu par Dorian Ginane
Le 27 mars 2025

Signature du Stagiaire :

Par Yann Bodiguel

9. Bibliographie

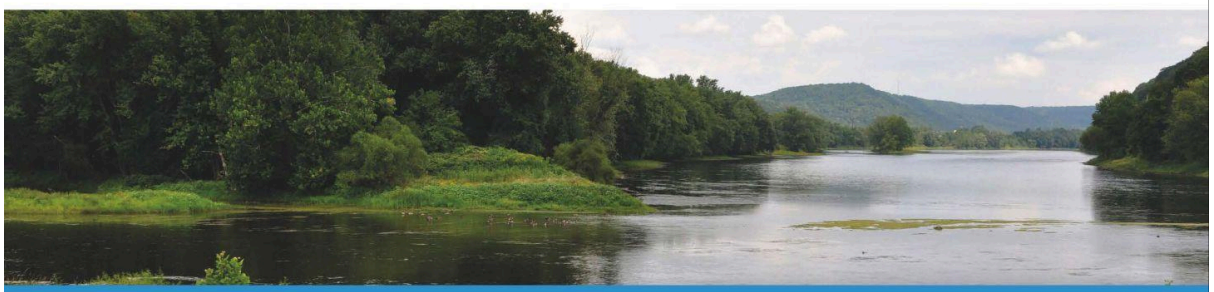
[1] ‘Raster graphics’, Wikipedia. Mar-2025. https://en.wikipedia.org/w/index.php?title=Raster_graphics
[2] ‘Displaying large amounts of markers as performant as possible · maplibre/maplibre-gl-js · Discussion #5207’, GitHub. . https://github.com/maplibre/maplibre-gl-js/discussions/5207
[3] ‘Create and style clusters - MapLibre GL JS’. . https://maplibre.org/maplibre-gl-js/docs/examples/cluster/
[4] ‘TileServer GL - Server for vector and raster maps with GL styles’. . https://openmaptiles.geo.data.gouv.fr/
[5] ‘TransformStyleFunction() - MapLibre GL JS’. . https://maplibre.org/maplibre-gl-js/docs/API/type-aliases/TransformStyleFunction/
[7] ‘Angular’. . https://angular.dev/tutorials
[8] ‘Become a ninja with Angular - the ebook’. . https://books.ninja-squad.com/angular

10. Annexes :

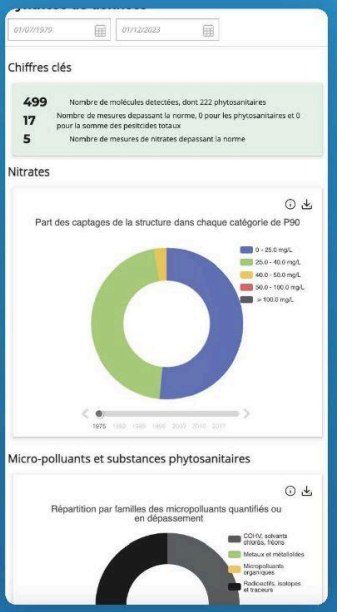
10.1 Fiche technique du produit Aqualit :

AQUALIT

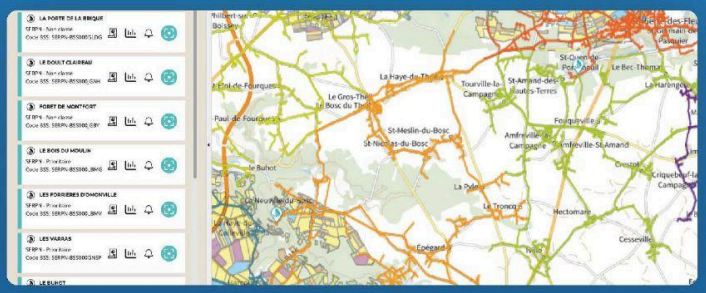
GEOMATYS



VOTRE PLATEFORME INTELLIGENTE D'ANALYSE DES EAUX



AQUALIT est un outil permettant de traiter de grandes quantités de données pour la gestion de la ressource en eau sur une plateforme SaaS facile d'utilisation. Grâce à AQUALIT, l'ensemble de vos données sont accessibles sur des graphiques faciles à intégrer, et contextualisés. Ajoutez des données complémentaires de votre territoire pour une analyse plus fine.



CHOISISSEZ la formule adaptée à vos besoins métiers....

... et **FAITES PARLER** vos données !

UN OUTIL PERFORMANT POUR L'ANALYSE DES MESURES SUR L'EAU

AQUALIT est une plateforme SaaS où n'importe quel producteur d'eau potable dispose d'un accès maitrisé à ses données pour faciliter l'exploitation des analyses et anticiper les évolutions du territoire pour la ressource en eau. La plateforme dispose d'une interface ergonomique et complète pour **l'analyse automatique** et le **traitement des données de qualité de l'eau**. Elle possède des outils intelligents, comme une fonctionnalité de notification en cas de dépassement de seuil, ou la possibilité de contextualiser géographiquement vos données.

Chiffres clés

10+
FORMATS DE DONNÉES GÉRÉS

4
MODULES

1000+
PARAMÈTRES

AQUALIT est le fruit d'une collaboration entre trois acteurs de l'eau qui ont constaté la nécessité d'accélérer et d'enrichir l'exploitation de leurs données d'analyse.

En 2022, Geomatys a été choisi pour le développement et la maintenance de la plateforme, résultat d'un dialogue entre les besoins métiers des acteurs et les capacités informatiques de Geomatys.

Depuis septembre 2023, cette plateforme, créée par Geomatys, est ouverte à d'autres acteurs de l'eau et continuera à évoluer avec de nouvelles fonctionnalités.

Les fonctionnalités d'AQUALIT

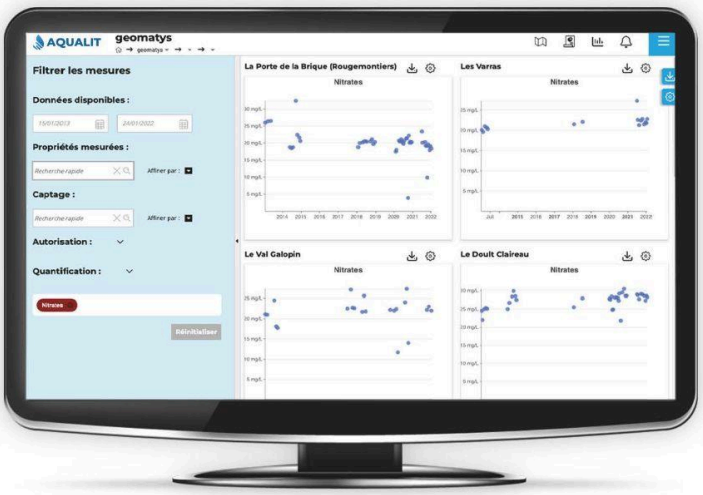
AQUALIT a été construit autour des référentiels métier (scénarios d'échange, référentiels paramètres du SANDRE, référentiels e-Phy)
Le développement de la plateforme a été réalisée autour de quatre grandes fonctionnalités, qui font d'AQUALIT un outil unique en son genre :

- Une **PLATEFORME GÉOGRAPHIQUE**, permettant de combiner vos mesures avec des informations liées au territoire administré,
- Un module d'**ANALYSE DES DONNÉES**, pour afficher et trier simplement vos données brutes,
- Un outil de **SYNTHÈSE DES MESURES**, afin d'agréger des résultats et d'établir des tendances sur la qualité de l'eau à l'échelle d'un territoire, d'une BAC d'un point de captage,
- Un **OUTIL D'ALERTE**, pour avertir les administrateurs en cas de dépassement du seuil d'une subtance donnée ou d'une variation suspecte d'un paramètre, afin de répondre aux enjeux des acteurs de l'eau.

LE MODULE D'ANALYSE DES MESURES

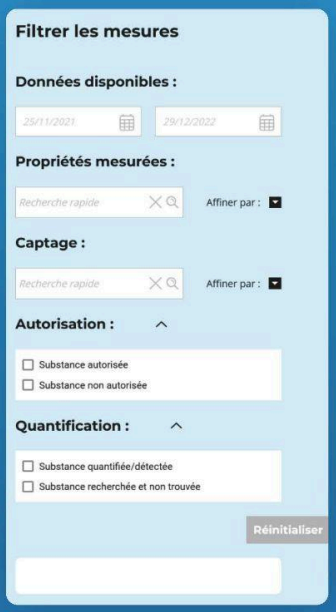
Le module d'analyse des mesures d'AQUALIT permet non seulement de lire simplement vos données de qualité de l'eau mais aussi de les traiter en quelques clics, pour obtenir rapidement des résultats et établir des tendances sur la qualité de l'eau de votre territoire.

Le module dispose de mécanismes d'affichage qui permettent de gérer avec fluidité des milliers de mesures.



Vous pouvez analyser vos données de qualité de l'eau en utilisant des filtres présents sur la plateforme, comme par exemple :

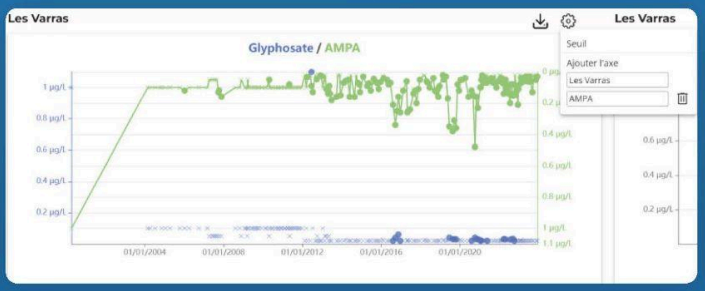
- **DONNÉES DISPONIBLES** : Filtre les mesures entre deux dates,
- **PROPRIÉTÉS MESURÉES** : Filtre les mesures sur les propriétés sélectionnées,
- **USAGE** : En fonction de l'usage attribué à une subtance (Base e-Phy) pour un utilisation agricole,
- **CAPTAGE** : filtre les mesures sur les captages sélectionnés,
- **AUTORISATION** : Filtre les substances à afficher en fonction de leur autorisation de mise sur le marché,
- **QUANTIFICATION** : Si la substance à été quatifiée au moins une fois dans l'intervalle de temps recherchée.

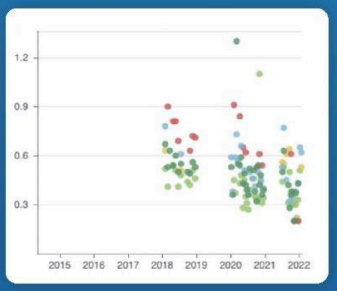


Le logiciel AQUALIT dispose de fonctionnalités d'import de données automatisées ou manuelles qui gèrent les formats de données du SANDRE et peut se connecter à multiples API ou divers superviseurs (SCADA)

Exemple de formats gérés :

- Scénarios d'échange du SANDRE,
- EDILABO,
- QESOUT,
- TOKAPI,
- ADES,
- NAÏADES,
- xls, csv....





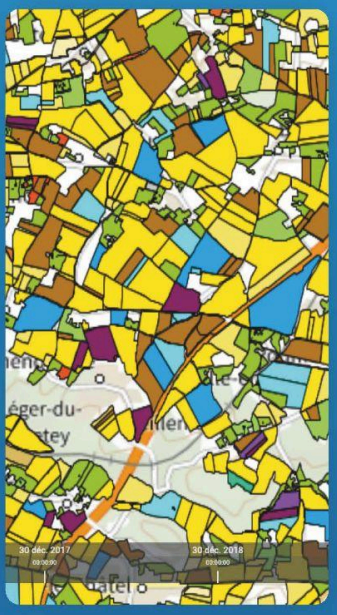
LE MODULE CARTOGRAPHIQUE

Le module cartographique d'AQUALIT vous permet de positionner geographiquement vos mesures avec les données contextuelles reliées à votre environnement (géologie, registre cadastral, etc..).

Il permet de visualiser les évolutions et les dynamiques de votre territoire dans le temps.

La visualisation cartographique proposée est capable de proposer une superposition des flux de données externes diffusés via les standards OGC avec des flux de données internes gérés par l'administrateur de la plateforme d'Aqualit.

Importez simplement vos données cartographiques



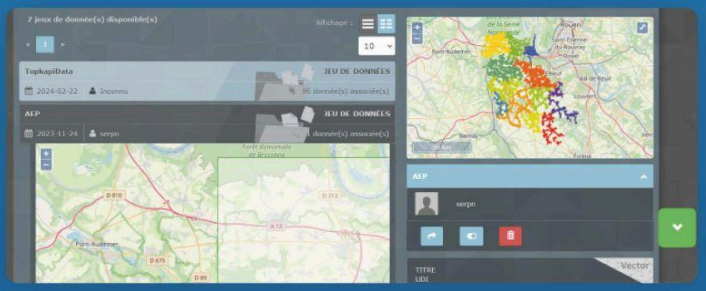
La forte expérience des équipes d' AQUALIT dans le domaine des systèmes d'infomation géographiques a permis de développer un support facilitant l'importation de données géospatiales.

Le serveur **EXAMIND** est un serveur de données cartographiques, il comprend donc des interfaces de chargement et de publication de couches cartographiques pour les administrateurs de la structure. Ses couches peuvent être ensuite configurées et publiées au niveau du portail.

Les administrateurs de données sont libres d'importer des données géographiques personnalisées au format Shp, GeoPackage, Geotiff... et configurer leurs styles et le diffuser via des services standards OGC

Pour les couches cartographiques, l'administrateur dispose d'un éditeur de styles qui lui permet de maîtriser la symbologie et la représentation des couches.

Les couches sont publiées au travers d'API cartographiques standard. Nous privilégions les nouvelles API OGC Rest qui favorisent leur réutilisation par des développeurs tiers. Les anciennes API demeurent disponibles.

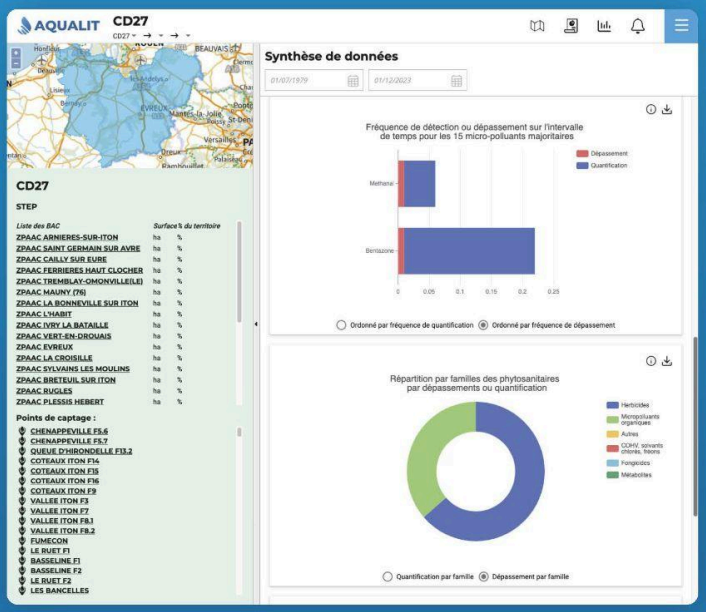


Si les couches ont une dimension temporelle (par exemple résultats des différents rapportages sur les masses d'eau...), elles sont publiées sous la forme d'un cube de données homogène afin de permettre une navigation cartographique dans le temps comme dans l'espace.

LE MODULE DE SYNTHÈSE

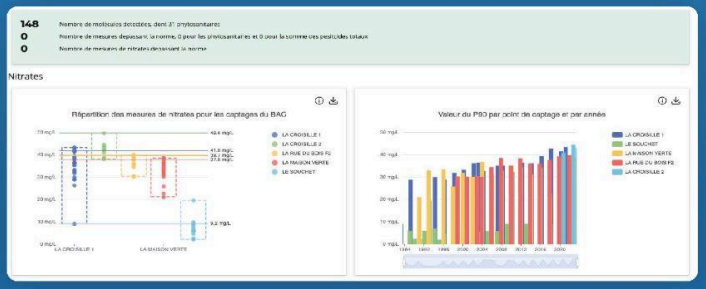
La page de synthèse d'AQUALIT est l'outil indispensable pour les exploitants de points de captage et les gestionnaires de la qualité de l'eau proposant une vue d'ensemble claire et précise de la qualité des relevés sur une temporalité modulable.

Grâce à ses fonctionnalités avancées, ce module de synthèse valorise toutes vos données brutes en informations exploitables, facilitant la prise de décision et la gestion proactive des ressources en eau.



3+
Niveaux d'agrégation géographique

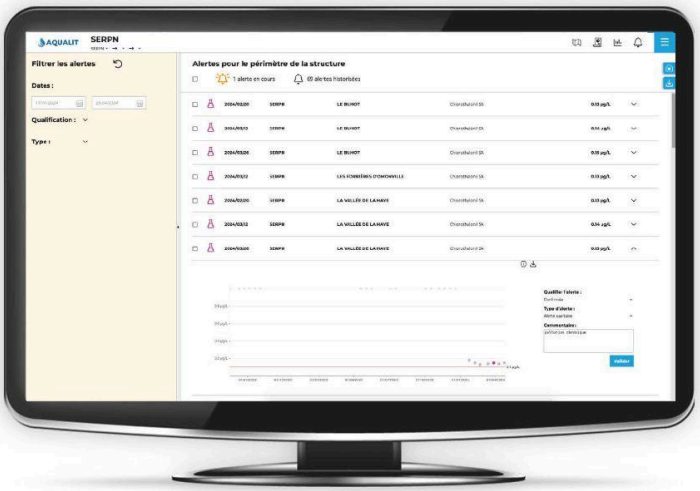
Les données de mesures sont agrégées selon différents niveaux de répartition géographique.
Pour mettre en place cette fonctionnalité nous utilisons des Shapefiles tels que les points localisant les prélèvements d'eau brute, les points des stations de productions, les polygones des AAC et des périmètres de protection des AEP.



L'outil de synthèse intègre des filtres temporels vous donnant la possibilité d'identifer facilement des tendances et d'observer leurs évolutions sur votre territoire.

LE MODULE D'ALERTE

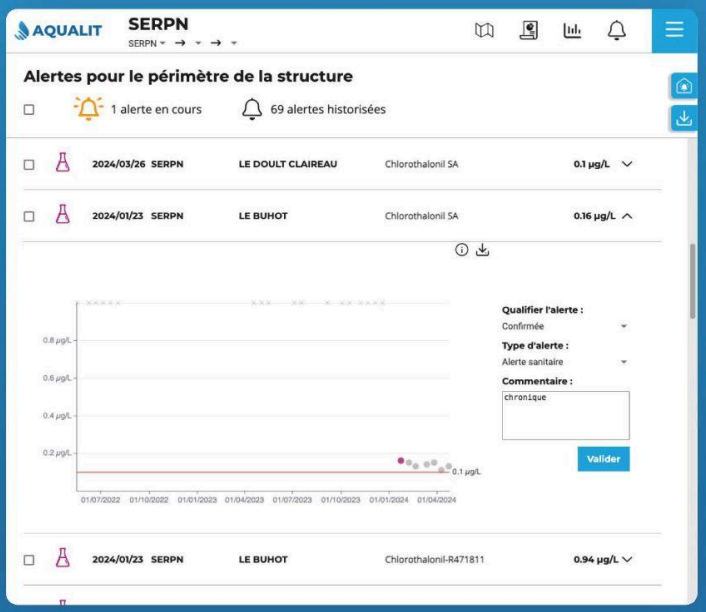
AQUALIT dispose d'un outil essentiel pour assurer la sécurité sanitaire : Le module d'alerte. En centralisant les alerte, en offrant des filtres personnalisables, et en permettant une gestion et une exportation facile des données, AQUALIT assure une surveillance efficace et une réactivité accrue face aux dépassements de seuils ou aux données suspectes.



L'outil dispose de deux types d'alertes :

- **Des alertes de validation** : lorsque la structure de la donnée est conforme mais que les valeurs sont suspicieuses au regard de seuil ou de variations définies,
- **Des alertes sanitaires** : lorsque la donnée dépasse un seuil réglementaire par exemple.

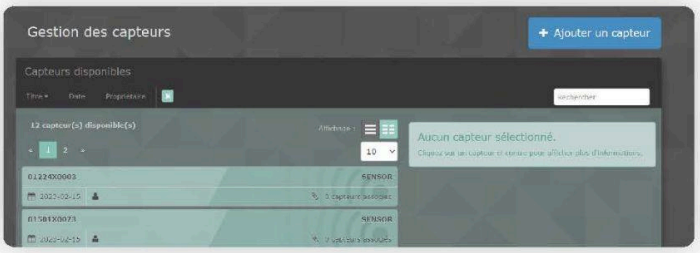
Le module permet à un utilisateur de s'abonner aux alertes d'un point de captage, d'un BAC ou de la structure entière, envoyant une **notification** en cas de dépassement de seuil d'une subtance.
Cette fonctionnalité garantit une réactivité maximale et une surveillance continue du territoire administré.



Les seuils d'AQUALIT sont définis à partir des seuils règlementaires du **SANDRE**, assurant une conformité aux normes nationales. Toutefois, ces seuils peuvent être modifiés par les administrateurs pour s'adapter aux contextes locaux spécifiques, offrant ainsi une flexibilité pour une surveillance personnalisée des ressources en eau.

POUR LES ADMINISTRATEURS

Chaque abonnement AQUALIT comprend plusieurs comptes administrateurs, qui gèrent les données et le paramétrage de la plateforme, de la gestion des seuils d'alertes au partage des données, en passant par la gestion des utilisateurs.



Gestion des rôles

AQUALIT offre une gestion avancée des utilisateurs et rôles. Les administrateurs peuvent créer des comptes, assigner des rôles spécifiques, et définir des permissions précises.

Les rôles varient de lecteur de données à administrateur complet. Cette flexibilité assure un accès sécurisé et adapté à chaque producteur d'eau potable.

- ADMINISTRATEUR :

l'utilisateur gère toutes les données et utilisateurs de sa structure, il accède par défaut à l'ensemble des données.

- MANAGER DES DONNÉES :

l'utilisateur peut gérer les données de sa structure mais pas les utilisateurs. Il a accès à l'ensemble des données de sa structure.

- USER :

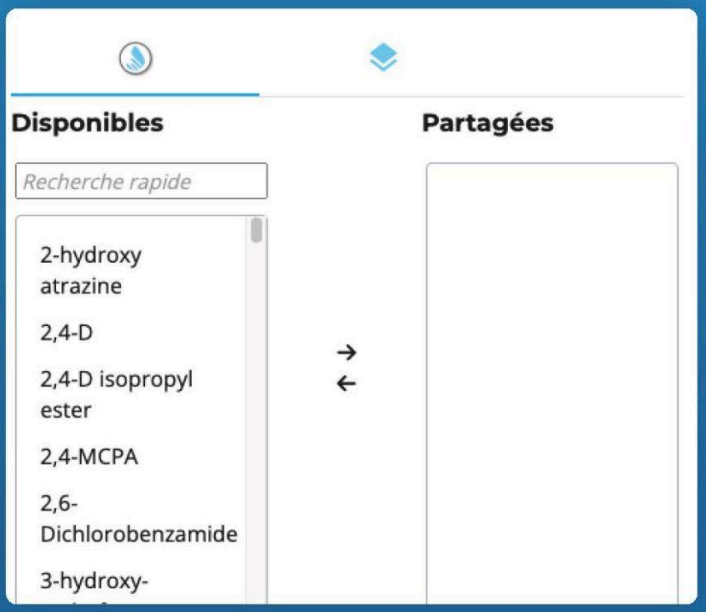
L'utilisateur peut consulter les données en fonction des droits qui lui sont accordés.

Partage des données

AQUALIT facilite le partage des données entre utilisateurs appartenant à des structures différentes. L'aministrateur peut donc partager des données d'analyse de son territoire à un partenaire sans avoir à exporter les données.

Gestion des données

Les administrateurs d'AQUALIT ont la possibilité d'importer leurs données de qualité des eaux ainsi que des données cartographiques, mais aussi de modifier les seuils d'alertes, le tout via l'interface Examind de la plateforme.



VOTRE ABONNEMENT AQUALIT

	<15 Captages	15 à 49 Captages	50 à 99 Captages	100+ Captages
Forfait déploiement	3000€	6000€	10 000€	
Préparation des données	Oui	Oui	Oui	
Formation utilisateurs	2 Utilisateurs	4 Utilisateurs	10 Utilisateurs	
Forfait annuel	2400€ / an	6000€ / an	10800€ / an	Nous contacter
Nombre d'utilisateurs	5	10	50	
Accès au support	Oui	Oui	Oui	
Nombre d'administrateurs	2	4	10	
Volume de données	30 Go	200 Go	600 Go	

L'abonnement

L'abonnement à AQUALIT est flexible et comporte plusieurs formules, adaptées en fonction du nombre de points de captage de votre structure. Si le territoire à administrer est conséquent (100+ captages), une offre personnalisée vous est proposée par les équipes d'AQUALIT.

Le support

Chaque abonnement à AQUALIT est accompagné d'un support client rapide et efficace de la part de Geomatys.

En cas de besoin spécifique, des compléments fonctionnels seront développés sur devis.

Le club utilisateur

Il consiste en un lieu d'échange pour les utilisateurs de la plateforme. Il permet de :

- partager les nouveautés réglementaires actuelles ou à venir,

- proposer des pistes d'amélioration de la plateforme qui peuvent ensuite être choisies pour être développées en tant que nouvelle fonctionnalité,

Il est animé par les clients historiques d'AQUALIT et dispose d'une inscription facultative à une liste de diffusion.

SUPPORT RAPIDE **ABONNEMENT MODULAIRE** **PLATEFORME COMMUNAUTAIRE**

PROJET D'INNOVATION EN PARTENARIAT AVEC

L'Eure en Normandie · Chartres Métropole · SERPN · eau seine Normandie

GEOMATYS

Arles (13), Montpellier (34)

contact@geomatys.com

Tél : +33 (0)4 84 49 02 26