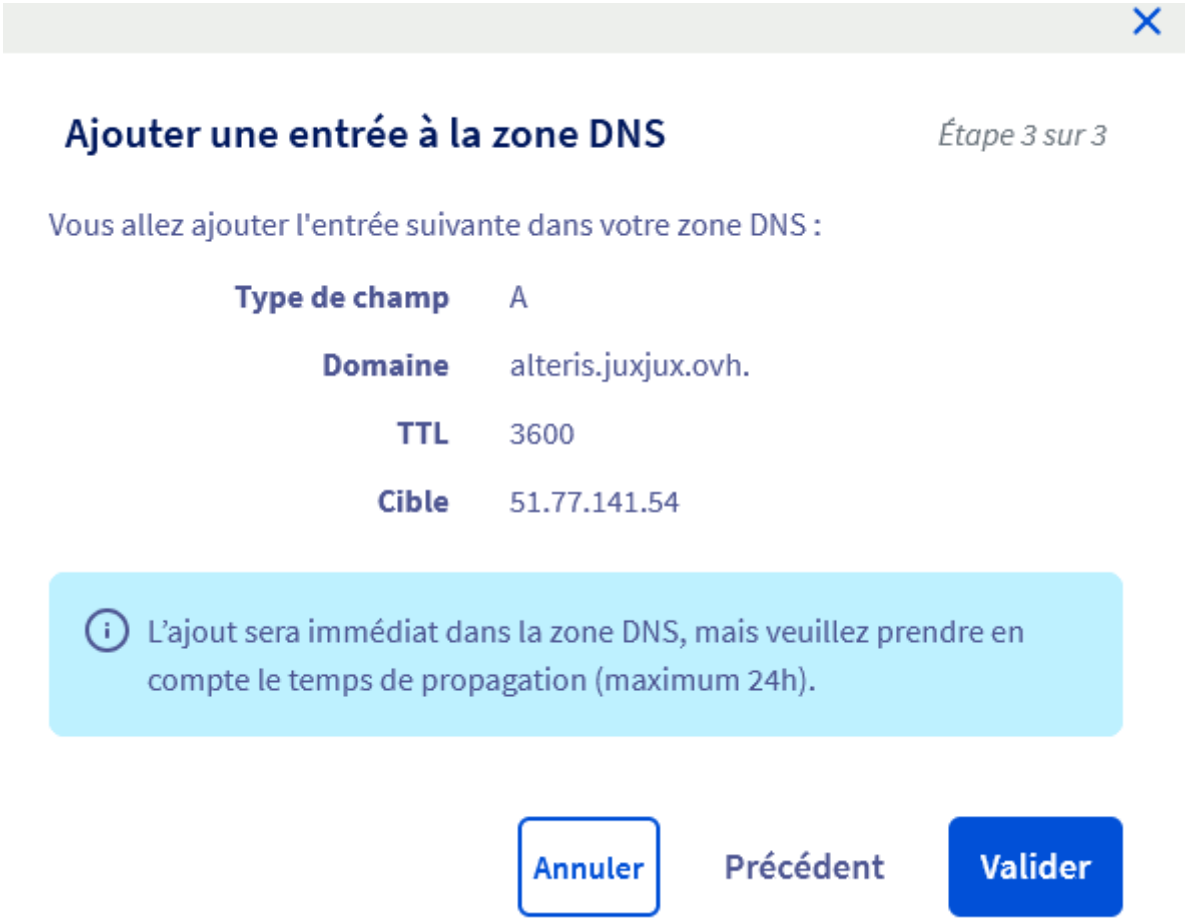
#A - METHODO - montage du websig

- Reverse Proxy - https://alteris.juxjux.ovh
- Mise en place de Node.js
- 260203 - installation PostgreSQL (la base de données) et PostGIS (l'extension qui ajoute les fonctions géographiques).
- 260205-Installation base de données Paris dans Alteris
- 260302 Mise en place du Frontend
- 260302 - Transformation du Frontend en page SIG
- 260302 - Integration des outils Leaflet et appel des couches SIG extérieures (raster notamment)
- 260302 - sauvegarde du projet Alteris (Code et bases de données)
- Analytique - créer des tables attributaires
- Intégration OSM
- osm2pgsql

Reverse Proxy - https://alteris.juxjux.ovh

Etape 1 - montage du reverse proxy



Configuration du proxy

```
server {
    listen 80;
    listen [::]:80;

    server_name alteris.juxjux.ovh;

    # Configuration des logs (optionnel mais recommandé pour le débogage)
    access_log /var/log/nginx/alteris_access.log;
    error_log /var/log/nginx/alteris_error.log;
```

```
    location / {
        # Redirection vers l'application Node.js sur le port 3000
        proxy_pass http://localhost:3000;

        # En-têtes essentiels pour que Node reçoive les bonnes infos
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_cache_bypass $http_upgrade;
    }
}
```

Création du lien symbolique

```
sudo ln -s /etc/nginx/sites-available/alteris-geomatic /etc/nginx/sites-enabled/
```

Vérification de la syntaxe

```
sudo nginx -t
```

Rechargement de NGINX

```
sudo systemctl reload nginx
```

Création du HTTPS

```
sudo certbot --nginx -d alteris.juxjux.ovh
```

Mise en place de Node.js

Étape 1 : Installation de Node.js

Copie et colle ces commandes l'une après l'autre :

1. **Récupérer le script d'installation officiel :**
   Bash
   ```
   curl -fsSL https://deb.nodesource.com/setup_20.x | sudo -E bash -
   ```
   (Tu vas voir beaucoup de lignes défiler, c'est normal, il prépare les dépôts).
2. **Installer Node.js :**
   Bash
   ```
   sudo apt install -y nodejs
   ```
3. **Installer les outils de compilation (par sécurité pour certains modules) :**
   Bash
   ```
   sudo apt install -y build-essential
   ```
4. **Vérification :** Vérifions que tout est là.
   Bash
   ```
   node -v
   npm -v
   ```
   Si tu vois `v20.x.x` s'afficher, c'est parfait.

Étape 2 : Préparation du dossier du projet

Maintenant que Node est là, nous pouvons créer l'environnement pour le projet Alteris.

1. **Créer le dossier (si ce n'est pas déjà fait) et entrer dedans :**
   Bash
   ```
   sudo mkdir -p /var/www/alteris-geomatic


   # On donne les droits à ton utilisateur (debian) pour pouvoir modifier les fichiers
   ```

```
sans 'sudo' à chaque fois
sudo chown -R $USER:$USER /var/www/alteris-geomatic


cd /var/www/alteris-geomatic
```

2. **Initialiser le projet Node.js :** Cette commande crée un fichier `package.json` qui sert de carte d'identité au projet.
   Bash
   ```
   npm init -y
   ```

3. **Installer les librairies nécessaires :** C'est ici qu'on installe les outils dont on a parlé :
   - `express` : Pour créer le serveur web.
   - `pg` : Pour que Node.js puisse parler à PostgreSQL.
   - `body-parser` : Pour lire les JSON envoyés.

   Bash
   ```
   npm install express pg body-parser
   ```

4. **Installer PM2 (Le gestionnaire de processus) :** C'est l'outil qui gardera ton serveur allumé même si tu te déconnectes.
   Bash
   ```
   sudo npm install -g pm2
   ```

Résumé de la situation

Maintenant, nous sommes vraiment prêts :

- [x] Base de données PostGIS installée.
- [x] Données test (Carré Paris) insérées dans la base.
- [x] Node.js installé.
- [x] Dossier projet créé et librairies installées.

Dis-moi quand c'est fait (vérifie juste avec `node -v` que tu as bien une version), et je te redonne le code des fichiers `db.js` et `server.js` pour qu'on finalise !

260203 - installation PostgreSQL (la base de données) et PostGIS (l'extension qui ajoute les fonctions géographiques).

# 1. Mettre à jour la liste des paquets
sudo apt update

 # 2. Installer Postgres et PostGIS
sudo apt install postgresql postgresql-contrib postgis -y

 # 3. Démarrer le service et s'assurer qu'il se lance au démarrage du serveur
sudo systemctl start postgresql
sudo systemctl enable postgresql

Étape 2 : Créer l'utilisateur et la Base de Données

Par défaut, PostgreSQL crée un utilisateur système nommé `postgres`. Nous allons l'utiliser pour créer ton propre utilisateur dédié au projet Alteris.

#1 Passe sur l'utilisateur administrateur de la base :

sudo -i -u postgres

#2 Crée ton utilisateur (rôle) :

createuser --interactive --pwprompt

Le système va te poser des questions. Réponds comme suit :

- **Enter name of role to add:** `alteris_user`

- **Enter password for new role:** (Choisis un mot de passe fort et **note-le**, tu en auras besoin pour le fichier `db.js`)
- **Enter it again:** (Confirme le mot de passe)
- **Shall the new role be a superuser?** `n` (Non)
- **Shall the new role be allowed to create databases?** `y` (Oui)
- **Shall the new role be allowed to create more new roles?** `n` (Non)

#3 **Crée la base de données :** Nous créons la base `alteris_db` et on dit qu'elle appartient à `alteris_user`.

createdb -O alteris_user alteris_db

#4 Activer la "Magie" Spatiale (PostGIS)

C'est l'étape cruciale. Pour l'instant, c'est une base classique. On doit lui greffer le module spatial. Toujours en tant que `postgres`, lance cette commande :

psql -d alteris_db -c "CREATE EXTENSION postgis;"

(Si tout va bien, il affichera simplement `CREATE EXTENSION`)
Ensuite, tu peux quitter l'utilisateur postgres pour revenir à ton utilisateur normal (root ou autre) :

260205-Installation base de données Paris dans Alteris

Allons peupler cette base de données. Nous allons créer une table pour stocker tes zones géographiques et y insérer notre fameux carré de test autour de Paris.

Connecte-toi à ton VPS et suis ces étapes :

Étape 1 : Connexion à la base de données

Nous allons nous connecter avec l'utilisateur que tu as créé (`alteris_user`).

1. Lance cette commande :
   Bash
   ```
   psql -U alteris_user -d alteris_db -h localhost
   ```
2. Saisis le mot de passe que tu as défini pour cet utilisateur. *(Le curseur ne bougera pas quand tu tapes, c'est normal sous Linux, valide avec Entrée).*

Tu devrais voir l'invite de commande changer pour : `alteris_db=>`

Étape 2 : Création de la table et insertion des données

Une fois connecté, copie et colle tout ce bloc de code d'un seul coup. Il va créer la table et ajouter la zone.

SQL
```
-- 1. On crée la table 'zones_interet'
CREATE TABLE zones_interet (
    id SERIAL PRIMARY KEY,
    nom VARCHAR(100),
    geom GEOMETRY(Polygon, 4326)
```

```
);

-- 2. On insère le carré autour de Paris
INSERT INTO zones_interet (nom, geom)
VALUES (
    'Zone Paris Test',
    ST_GeomFromText('POLYGON((2.25 48.80, 2.45 48.80, 2.45 48.90, 2.25 48.90, 2.25 48.80))',
4326)
);
```

Étape 3 : Vérification

Pour être sûr que la donnée est bien enregistrée, tape cette commande :

SQL

```
SELECT nom, ST_AsText(geom) FROM zones_interet;
```

Si tu vois une ligne avec "Zone Paris Test" et une suite de coordonnées, c'est gagné !

Tu peux ensuite quitter l'interface SQL en tapant :

SQL

```
\q
```

Résumé de l'état actuel :

- [x] Base PostGIS prête.
- [x] Données de test insérées (si tu as fait l'étape ci-dessus).
- [x] Node.js installé (lors de l'étape précédente).

Si tu as bien réussi l'insertion, nous pouvons passer à la **création des fichiers du serveur (Code)**. Tu me confirmes ?

C'est parfait ! Tu as maintenant :

1. ✅ Le serveur web (Nginx).
2. ✅ Le moteur Node.js.
3. ✅ La base de données spatiale avec ta zone de test.

Nous sommes dans la dernière ligne droite : **le codage de l'application**.

Tu es déjà dans le bon dossier (`/var/www/alteris-geomatic`). Nous allons créer deux fichiers.

Étape 1 : Le fichier de connexion (`db.js`)

Ce fichier permet à Node.js d'ouvrir la porte de ta base de données.

1. Crée le fichier :
   Bash
   ```
   nano db.js
   ```
2. Copie-colle le code suivant. ⚠ **IMPORTANT :** Remplace `'METTRE_TON_MOT_DE_PASSE_ICI'` par le vrai mot de passe que tu as défini pour `alteris_user`.
   JavaScript
   ```
   const { Pool } = require('pg');

   const pool = new Pool({
     user: 'alteris_user',
     host: 'localhost',
     database: 'alteris_db',
     password: 'METTRE_TON_MOT_DE_PASSE_ICI',
     port: 5432,
   });

   module.exports = pool;
   ```
3. Sauvegarde et quitte : `Ctrl + O`, `Entrée`, `Ctrl + X`.

Étape 2 : Le fichier Serveur (`server.js`)

C'est le cerveau. Il reçoit le point GPS, demande à PostGIS s'il est dans la zone, et renvoie la réponse.

1. Crée le fichier :
   Bash
   ```
   nano server.js
   ```
2. Copie-colle ce code complet :
   JavaScript
   ```
   const express = require('express');
   const bodyParser = require('body-parser');
   const pool = require('./db'); // On importe ta connexion db.js

   const app = express();
   const PORT = 3000;

   app.use(bodyParser.json());

   // Route de vérification simple
   app.get('/', (req, res) => {
       res.send('Serveur Alteris Géomatique opérationnel □□');
   });

   // --- LE CŒUR DU SYSTÈME ---
   app.post('/api/check-zone', async (req, res) => {
     try {
       const geoJsonInput = req.body;
       console.log("□□Requête reçue :", JSON.stringify(geoJsonInput));

       // Vérification de sécurité
       if (!geoJsonInput || !geoJsonInput.coordinates) {
         return res.status(400).json({
             error: "Format invalide.",
             exemple: {type: 'Point', coordinates: [2.35, 48.85]}
         });
       }

       // Requête SQL vers PostGIS
       // ST_Contains renvoie TRUE si la geometry contient le point
   ```

```
    const query = `
      SELECT nom
      FROM zones_interet
      WHERE ST_Contains(
        geom,
        ST_SetSRID(ST_GeomFromGeoJSON($1), 4326)
      );
    `;

    const result = await pool.query(query, [JSON.stringify(geoJsonInput)]);

    // Réponse au client
    if (result.rows.length > 0) {
      res.json({
        status: "success",
        message: `Le point est DANS la zone : ${result.rows[0].nom}`,
        in_zone: true,
        zone: result.rows[0]
      });
    } else {
      res.json({
        status: "success",
        message: "Le point est HORS de toute zone connue.",
        in_zone: false
      });
    }

  } catch (err) {
    console.error("Erreur serveur :", err);
    res.status(500).json({ error: "Erreur interne du serveur" });
  }
});

// Démarrage
app.listen(PORT, () => {
  console.log(`□□Application lancée sur le port ${PORT}`);
});
```

3. Sauvegarde et quitte : `Ctrl + O`, `Entrée`, `Ctrl + X`.

Étape 3 : Allumage des moteurs (PM2)

On lance l'application en tâche de fond.

Bash

```
# Lancer le serveur
pm2 start server.js --name alteris-api

# Sauvegarder pour qu'il redémarre si le VPS reboote
pm2 save
pm2 startup
```

Étape 4 : Le test final

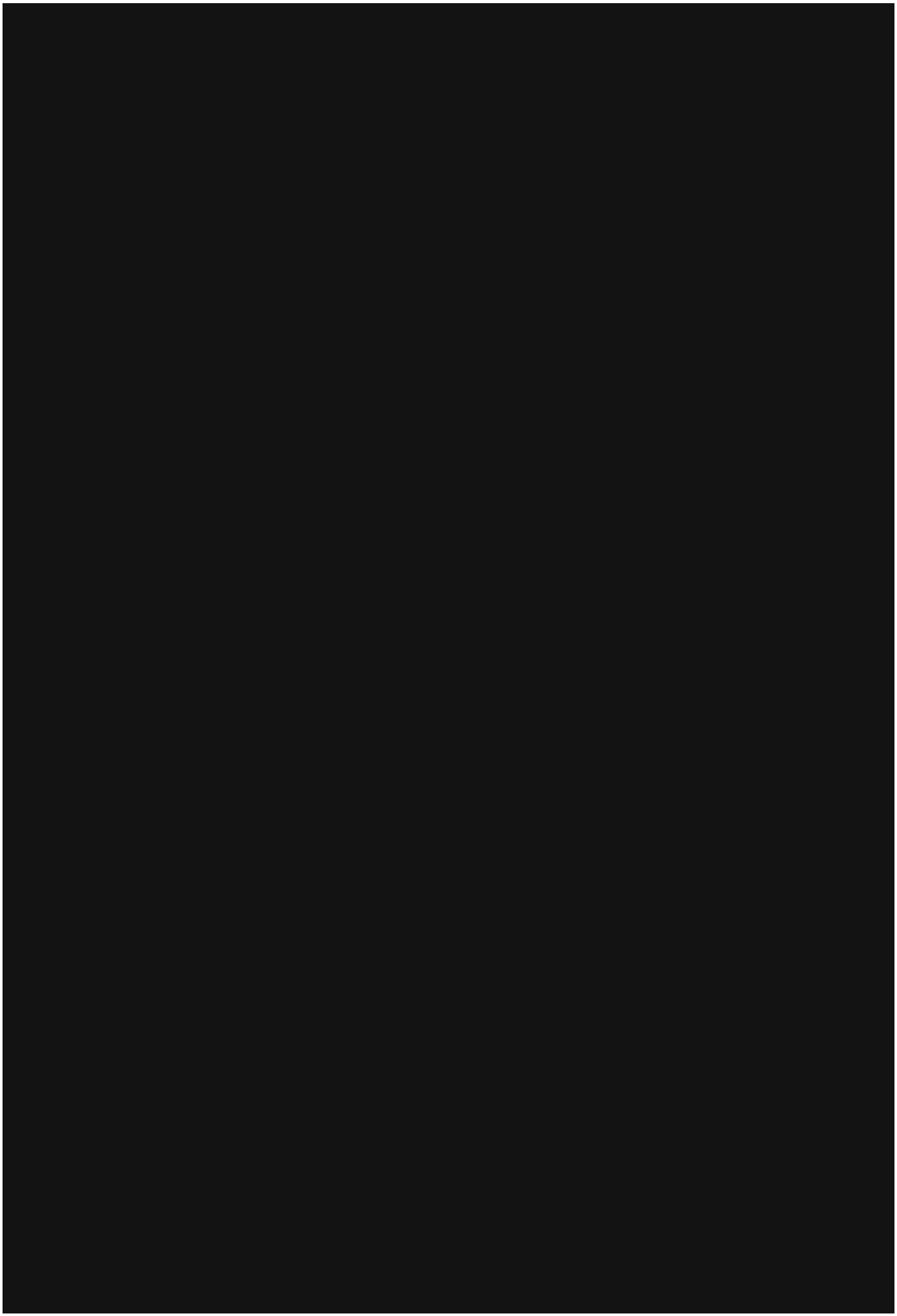
C'est le moment de vérité ! Testons depuis ton VPS (ou ton PC) si tout fonctionne via ton nom de domaine.

Test positif (Dans Paris) :

Bash

```
curl -X POST https://alteris.juxjux.ovh/api/check-zone \
-H "Content-Type: application/json" \
-d '{"type": "Point", "coordinates": [2.35, 48.85]}'
```

Si tu reçois : `"message": "Le point est DANS la zone : Zone Paris Test"`, c'est gagné ! Dis-moi le résultat !



C'est limpide ! Merci pour cette liste. Tu as une sacrée installation Docker (Immich, Jellyfin, Komga... c'est du propre !).

On voit clairement le coupable à la ligne 16 : `grafana 0.0.0.0:3000->3000/tcp`

Il monopolise le port 3000. Par contre, le port **3001** est libre (personne ne l'utilise dans ta liste).

Nous allons donc déménager l'application Alteris sur le port **3001**. Voici les 3 étapes précises pour régler le conflit.

Étape 1 : Modifier le port dans Node.js

1. Ouvre ton fichier serveur :

   Bash

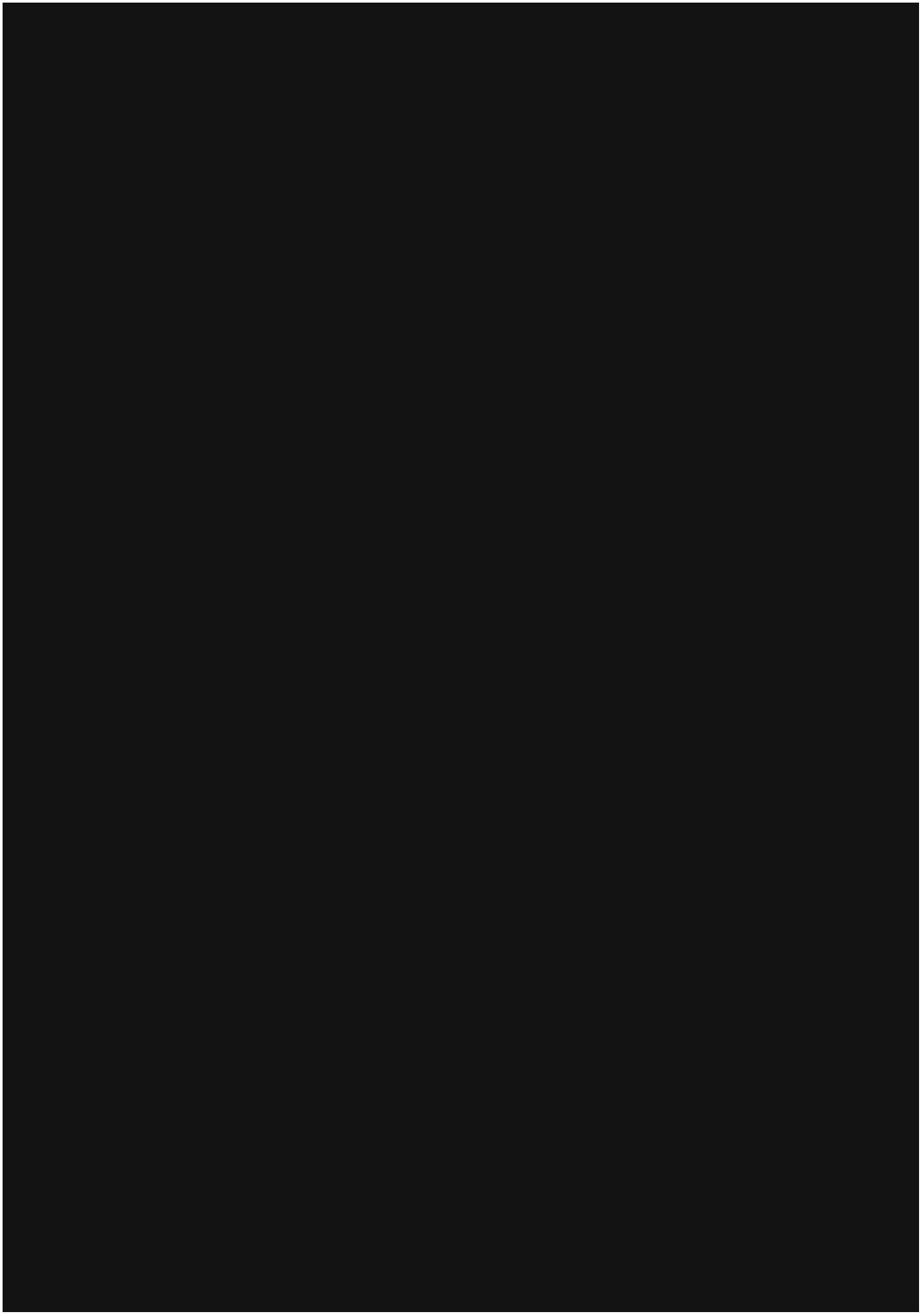
   ```
   nano /var/www/alteris-geomatic/server.js
   ```

2. Tout en haut du fichier, change la ligne `const PORT = 3000;` par :

   JavaScript

   ```
   const PORT = 3001;
   ```

3. Sauvegarde (`Ctrl+O`, `Entrée`) et quitte (`Ctrl+X`).



Étape 2 : Appliquer le changement (PM2)

Pour que Node.js prenne en compte la modification, il faut redémarrer le processus.

Bash

```
pm2 restart alteris-api
```

Étape 3 : Mettre à jour le portier (Nginx)

Il faut dire à Nginx d'envoyer les visiteurs vers le 3001 au lieu du 3000 (qui est Grafana).

1. Ouvre la config du site :

   Bash

   ```
   sudo nano /etc/nginx/sites-available/alteris-geomatic
   ```

2. Cherche la ligne `proxy_pass` et modifie-la ainsi :

   Nginx

   ```
   proxy_pass http://localhost:3001;
   ```

3. Sauvegarde et quitte (`Ctrl+O`, `Entrée`, `Ctrl+X`).

4. Recharge Nginx :

   Bash

```
sudo systemctl reload nginx
```

Le Test Final (Libéré de Grafana)

Maintenant, Grafana restera sur le port 3000, et ton application géomatique vivra sa vie sur le port 3001, accessible via ton nom de domaine.

Relance la commande de test :

Bash

```
curl -X POST https://alteris.juxjux.ovh/api/check-zone \
-H "Content-Type: application/json" \
-d '{"type": "Point", "coordinates": [2.35, 48.85]}'
```

Bravo ! □□ C'est une réussite totale.

Regarde ta réponse : `{"in_zone":true,"message":"Le point est DANS la zone..."}`.

Cela confirme que **toute la chaîne fonctionne** :

1. **Nginx** a bien reçu la requête sécurisée (HTTPS).
2. Il l'a passée à **Node.js** sur le port 3001.
3. Node.js a interrogé **PostGIS**.
4. PostGIS a fait le calcul géométrique ("Ce point est-il dans le carré ?").
5. Le résultat est remonté jusqu'à toi.

Ton backend géomatique est opérationnel.

260302 Mise en place du Frontend

Nous allons créer une page web contenant une carte interactive (basée sur OpenStreetMap). Quand tu cliqueras n'importe où sur cette carte, la page web enverra les coordonnées à ton API (que nous avons codée la semaine dernière). Ton serveur vérifiera si le clic est dans la zone "Paris Test" et affichera le résultat directement sur la carte sous forme de petite bulle (popup).

Pour cela, nous allons utiliser **Leaflet.js**, qui est la bibliothèque standard, légère et gratuite pour faire de la cartographie sur le web.

Voici les 3 étapes pour y parvenir.

Étape 1 : Créer le dossier "public"

Sur ton VPS, nous allons créer un dossier qui contiendra les fichiers visibles par les utilisateurs (comme les pages HTML).

1. Connecte-toi à ton VPS et place-toi dans le dossier du projet :
   Bash
   ```
   cd /var/www/alteris-geomatic
   ```
2. Crée le dossier `public` :
   Bash
   ```
   mkdir public
   ```

Étape 2 : Créer la page web (`index.html`)

C'est ici que nous allons intégrer la carte.

1. Crée et ouvre le fichier HTML dans le dossier `public` :
   Bash
   ```
   nano public/index.html
   ```

2. Copie et colle ce code complet. J'ai ajouté des commentaires pour t'expliquer exactement ce que fait chaque partie :
   HTML

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Alteris - Carte Interactive</title>

    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.4/dist/leaflet.css" />

    <style>
        /* On s'assure que la carte prenne tout l'écran */
        body { margin: 0; padding: 0; }
        #maCarte { width: 100vw; height: 100vh; }
    </style>
</head>
<body>

    <div id="maCarte"></div>

    <script src="https://unpkg.com/leaflet@1.9.4/dist/leaflet.js"></script>

    <script>
        // --- INITIALISATION DE LA CARTE ---
        // On centre la vue sur Paris avec un niveau de zoom de 10
        const map = L.map('maCarte').setView([48.85, 2.35], 10);

        // On ajoute le fond de plan gratuit OpenStreetMap
        L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
            maxZoom: 19,
            attribution: '© OpenStreetMap contributors'
        }).addTo(map);

        // --- LOGIQUE DE CLIC ---
        // On écoute l'événement "click" sur la carte
        map.on('click', async function(evenement) {
```

```
            // On récupère la latitude et la longitude cliquées
            const lat = evenement.latlng.lat;
            const lng = evenement.latlng.lng;

            // On prépare notre format GeoJSON pour l'API
            // Attention : en GeoJSON, on met toujours [Longitude, Latitude]
            const pointGeoJSON = {
                type: "Point",
                coordinates: [lng, lat]
            };

            try {
                // On envoie la donnée à notre serveur Node.js
                const reponse = await fetch('/api/check-zone', {
                    method: 'POST',
                    headers: { 'Content-Type': 'application/json' },
                    body: JSON.stringify(pointGeoJSON)
                });

                const resultat = await reponse.json();

                // On affiche le message du serveur dans une bulle sur la carte
                L.popup()
                    .setLatLng(evenement.latlng)
                    .setContent("<b>Résultat spatial :</b><br>" + resultat.message)
                    .openOn(map);

            } catch (erreur) {
                console.error("Erreur :", erreur);
                alert("Impossible de joindre le serveur Alteris.");
            }
        });
    </script>
</body>
</html>
```

3. Sauvegarde et quitte (`Ctrl+O`, `Entrée`, `Ctrl+X`).

Étape 3 : Indiquer au serveur Node.js d'afficher cette page

Actuellement, notre serveur `server.js` ne sait faire que de l'API. Nous allons lui dire de "servir" notre nouveau dossier public lorsqu'un visiteur arrive sur le site.

1. Ouvre ton fichier serveur :
   Bash
   ```
   nano server.js
   ```
2. Trouve ces lignes au début de ton fichier :
   JavaScript
   ```
   app.use(bodyParser.json());

   // Route de vérification simple
   app.get('/', (req, res) => {
       res.send('Serveur Alteris Géomatique opérationnel □□');
   });
   ```
3. **Remplace la route** `app.get(...)` par cette simple ligne de code qui indique le dossier public :
   JavaScript
   ```
   app.use(bodyParser.json());

   // On demande à Express de servir les fichiers du dossier "public"
   app.use(express.static('public'));
   ```
4. Sauvegarde et quitte (`Ctrl+O`, `Entrée`, `Ctrl+X`).

Étape 4 : Redémarrer et Tester

Pour que Node.js lise ton fichier `server.js` mis à jour, il faut le relancer via PM2 :

Bash
```
pm2 restart alteris-api
```

C'est l'heure du test visuel ! □□

Ouvre ton navigateur web (Chrome, Firefox, etc.) et rends-toi sur l'adresse de ton projet : □□

https://alteris.juxjux.ovh

Tu devrais voir une magnifique carte s'afficher.

260302 - Transformation du Frontend en page SIG

Maintenant, il ne nous reste plus qu'à s'assurer que ton fichier **`index.html`** (le frontend) est prêt à réceptionner ces données pour les dessiner.

Voici le code complet et propre pour ton fichier `index.html`. Il contient la carte, le chargement automatique des zones au démarrage, et la gestion du clic.

Mise à jour de `index.html`

1. Ouvre le fichier :
   Bash

   ```
   nano /var/www/alteris-geomatic/public/index.html
   ```

2. Remplace tout le contenu par celui-ci :

HTML

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Alteris - SIG Interactif</title>

    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.4/dist/leaflet.css" />

    <style>
        body { margin: 0; padding: 0; }
        #maCarte { width: 100vw; height: 100vh; background: #f0f0f0; }
        .info-panel {
            position: absolute; top: 10px; right: 10px; z-index: 1000;
            background: white; padding: 10px; border-radius: 5px;
```

```
            box-shadow: 0 0 15px rgba(0,0,0,0.2); font-family: sans-serif;
        }
    </style>
</head>
<body>

    <div class="info-panel">
        <b>Alteris Geomatic</b><br>
        Cliquez sur la carte pour tester.
    </div>
    <div id="maCarte"></div>

    <script src="https://unpkg.com/leaflet@1.9.4/dist/leaflet.js"></script>

    <script>
        // 1. Initialisation de la carte sur Paris
        const map = L.map('maCarte').setView([48.85, 2.35], 11);

        // 2. Ajout du fond de plan OpenStreetMap
        L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
            attribution: '© OpenStreetMap'
        }).addTo(map);

        // 3. Fonction pour charger et dessiner les zones depuis l'API
        async function chargerZones() {
            try {
                const reponse = await fetch('/api/zones');
                const data = await reponse.json();

                L.geoJSON(data, {
                    style: {
                        color: '#e67e22',   // Orange Alteris
                        weight: 3,
                        fillOpacity: 0.3
                    },
                    onEachFeature: function (feature, layer) {
                        layer.bindPopup("<b>Zone :</b> " + feature.properties.nom);
                    }
                }).addTo(map);
                console.log("Zones chargées avec succès");
```

```
            } catch (err) {
                console.error("Erreur chargement zones:", err);
            }
        }

        // 4. Gestion du clic pour vérification spatiale
        map.on('click', async function(e) {
            const point = {
                type: "Point",
                coordinates: [e.latlng.lng, e.latlng.lat]
            };

            try {
                const reponse = await fetch('/api/check-zone', {
                    method: 'POST',
                    headers: { 'Content-Type': 'application/json' },
                    body: JSON.stringify(point)
                });
                const res = await reponse.json();

                L.popup()
                    .setLatLng(e.latlng)
                    .setContent(res.message)
                    .openOn(map);
            } catch (err) {
                console.log("Erreur lors du clic:", err);
            }
        });

        // Lancement au chargement
        chargerZones();
    </script>
</body>
</html>
```

Finalisation

1. Sauvegarde avec `Ctrl+O`, `Entrée` et quitte avec `Ctrl+X`.
2. Tu n'as pas besoin de redémarrer PM2 pour un changement de fichier HTML (Nginx/Express le servent en direct), mais fais-le par sécurité si tu as un doute :
   Bash
   ```
   pm2 restart alteris-api
   ```

Le test ultime

Rends-toi sur **https://alteris.juxjux.ovh**.

Normalement :

- Le carré de Paris apparaît **immédiatement** en orange dès que la page s'ouvre.
- Si tu cliques **dedans**, la bulle confirme la présence.
- Si tu cliques **dehors**, elle confirme l'absence.

Est-ce que le tracé orange est enfin visible sur ton navigateur ? Si oui, **souhaites-tu que je t'apprenne comment insérer de nouvelles zones dans la base sans passer par la ligne de commande ?**

260302 - Integration des outils Leaflet et appel des couches SIG extérieures (raster notamment)

C'est une excellente nouvelle ! La boucle est bouclée : ton serveur lit la base de données, l'API transmet les coordonnées, et ton navigateur dessine le résultat.

Pour passer à la vitesse supérieure, nous allons ajouter la capacité de **créer de nouvelles zones directement depuis la carte**. L'idée est d'utiliser un petit outil de dessin : tu dessines un polygone, tu lui donnes un nom, et "paf", il s'enregistre dans PostGIS.

Voici le plan :

1. **Côté Serveur** : Créer une route `POST /api/save-zone` pour recevoir le nouveau polygone.
2. **Côté Frontend** : Ajouter la barre d'outils **Leaflet.draw** pour dessiner sur la carte.

Étape 1 : La route d'enregistrement (`server.js`)

Ouvre ton fichier `server.js` et ajoute cette route juste avant le `app.listen` :

JavaScript

```
// --- SAUVEGARDER UNE NOUVELLE ZONE ---
app.post('/api/save-zone', async (req, res) => {
  try {
    const { nom, geojson } = req.body;
```

```
    // Requête SQL pour insérer le polygone
    // ST_GeomFromGeoJSON convertit le JSON du navigateur en géométrie PostGIS
    const query = `
      INSERT INTO zones_interet (nom, geom)
      VALUES ($1, ST_SetSRID(ST_GeomFromGeoJSON($2), 4326))
      RETURNING id;
    `;

    const result = await pool.query(query, [nom, JSON.stringify(geojson.geometry)]);

    res.json({ status: "success", message: "Zone enregistrée !", id: result.rows[0].id });
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: "Erreur lors de la sauvegarde" });
  }
});
```

N'oublie pas de redémarrer avec `pm2 restart alteris-api` après avoir sauvegardé.

Étape 2 : L'interface de dessin (`index.html`)

Nous allons ajouter la bibliothèque **Leaflet.draw** dans ton fichier HTML.

1. Ouvre `public/index.html`.
2. Ajoute les liens CSS/JS de Leaflet.draw dans le `<head>` et avant la fin du `<body>`.
3. Ajoute la logique de sauvegarde.

Voici la partie du code à modifier/ajouter dans ton script :

HTML

```
<link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/leaflet.draw/1.0.4/leaflet.draw.css"/>

<script
```

```
src="https://cdnjs.cloudflare.com/ajax/libs/leaflet.draw/1.0.4/leaflet.draw.js"></script>

<script>
    // ... (ton code existant) ...

    // 5. Configuration de l'outil de dessin
    const drawnItems = new L.FeatureGroup();
    map.addLayer(drawnItems);

    const drawControl = new L.Control.Draw({
        edit: { featureGroup: drawnItems },
        draw: {
            polygon: true,   // On autorise les polygones
            polyline: false,
            rectangle: true,
            circle: false,
            marker: false
        }
    });
    map.addControl(drawControl);

    // 6. Action quand le dessin est terminé
    map.on(L.Draw.Event.CREATED, async function (e) {
        const layer = e.layer;
        const geojson = layer.toGeoJSON();

        const nomZone = prompt("Nom de cette nouvelle zone ?");
        if (!nomZone) return;

        // Envoi au serveur
        try {
            const reponse = await fetch('/api/save-zone', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify({ nom: nomZone, geojson: geojson })
            });
            const res = await reponse.json();

            if(res.status === "success") {
                alert("Zone '" + nomZone + "' enregistrée avec succès !");
```

```
                location.reload(); // On rafraîchit pour voir la nouvelle zone orange
            }
        } catch (err) {
            alert("Erreur lors de la sauvegarde.");
        }
    });
</script>
```

Comment tester ?

1. Rafraîchis ta page.
2. Tu verras une nouvelle **barre d'outils à gauche**.
3. Clique sur l'icône **Polygone**, dessine une forme sur la carte et ferme-la en cliquant sur le premier point.
4. Entre un nom (ex: "Zone Test 2").
5. La page va se recharger et ta nouvelle zone apparaîtra en orange, prête à être interrogée par un clic !

Final du server-.js

```
const express = require('express');
const bodyParser = require('body-parser');
const pool = require('./db');

const app = express();
const PORT = 3001;

app.use(bodyParser.json());
app.use(express.static('public'));

// 1. RÉCUPÉRER TOUTES LES ZONES
app.get('/api/zones', async (req, res) => {
  try {
    const query = `SELECT id, nom, ST_AsGeoJSON(geom) as geometrie FROM zones_interet;`;
```

```
    const result = await pool.query(query);
    const features = result.rows.map(row => ({
      type: "Feature",
      properties: { nom: row.nom },
      geometry: JSON.parse(row.geometrie)
    }));
    res.json({ type: "FeatureCollection", features: features });
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: "Erreur chargement zones" });
  }
});

// 2. VÉRIFIER SI UN CLIC EST DANS UNE ZONE
app.post('/api/check-zone', async (req, res) => {
  try {
    const geoJsonInput = req.body;
    const query = `
      SELECT nom FROM zones_interet
      WHERE ST_Contains(geom, ST_SetSRID(ST_GeomFromGeoJSON($1), 4326));
    `;
    const result = await pool.query(query, [JSON.stringify(geoJsonInput)]);
    if (result.rows.length > 0) {
      res.json({ status: "success", message: `Le point est DANS : ${result.rows[0].nom}` });
    } else {
      res.json({ status: "success", message: "Le point est HORS zone." });
    }
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: "Erreur calcul spatial" });
  }
});

// 3. SAUVEGARDER UNE NOUVELLE ZONE
app.post('/api/save-zone', async (req, res) => {
  try {
    const { nom, geojson } = req.body;
    const query = `
      INSERT INTO zones_interet (nom, geom)
      VALUES ($1, ST_SetSRID(ST_GeomFromGeoJSON($2), 4326))
      RETURNING id;
    `;
    const result = await pool.query(query, [nom, JSON.stringify(geojson.geometry)]);
    res.json({ status: "success", id: result.rows[0].id });
  } catch (err) {
    console.error(err);
```

```
    res.status(500).json({ error: "Erreur sauvegarde" });
  }
});

app.listen(PORT, () => {
  console.log(`□□ Serveur Alteris prêt sur le port ${PORT}`);
});
```

Final du frontend - index.html

avec appel des images aériennes de Google (IGN par la suite)

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Alteris - SIG Expert</title>

    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.4/dist/leaflet.css" />
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/leaflet.draw/1.0.4/leaflet.draw.css"/>

    <style>
        body { margin: 0; padding: 0; font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif; }
        #maCarte { width: 100vw; height: 100vh; background: #222; }

        .info-panel {
            position: absolute; top: 10px; right: 10px; z-index: 1000;
            background: rgba(255, 255, 255, 0.9); padding: 15px; border-radius: 8px;
            box-shadow: 0 0 15px rgba(0,0,0,0.3); max-width: 250px;
        }
        h3 { margin: 0 0 5px 0; color: #e67e22; }
        p { margin: 0; font-size: 0.9em; color: #333; }
    </style>
</head>
<body>

    <div class="info-panel">
        <h3>Alteris Geomatic</h3>
```

```
    <p><b>Dessin :</b> Utilisez les outils à gauche.</p>
    <p><b>Test :</b> Cliquez sur la carte pour vérifier si vous êtes dans une zone.</p>
</div>

<div id="maCarte"></div>

<script src="https://unpkg.com/leaflet@1.9.4/dist/leaflet.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/leaflet.draw/1.0.4/leaflet.draw.js"></script>

<script>
    // 1. DÉFINITION DES FONDS DE CARTE
    const osm = L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
        attribution: '© OpenStreetMap'
    });

    const googleSat = L.tileLayer('http://{s}.google.com/vt/lyrs=s&x={x}&y={y}&z={z}', {
        maxZoom: 20,
        subdomains:['mt0','mt1','mt2','mt3'],
        attribution: '© Google Satellite'
    });

    // 2. INITIALISATION DE LA CARTE (Satellite par défaut)
    const map = L.map('maCarte', {
        center: [48.85, 2.35],
        zoom: 12,
        layers: [googleSat]
    });

    // 3. SÉLECTEUR DE COUCHES
    const baseMaps = {
        "Satellite (Google)": googleSat,
        "Plan (OSM)": osm
    };
    L.control.layers(baseMaps).addTo(map);

    // 4. GESTION DES DESSINS (Stockage temporaire à l'écran)
    const drawnItems = new L.FeatureGroup();
    map.addLayer(drawnItems);

    const drawControl = new L.Control.Draw({
        edit: { featureGroup: drawnItems },
        draw: {
            polygon: { allowIntersection: false, showArea: true },
            rectangle: true,
            polyline: false, circle: false, marker: false, circlemarker: false
        }
```

```
});
map.addControl(drawControl);

// 5. CHARGEMENT DES ZONES DEPUIS POSTGIS
async function chargerZones() {
    try {
        const reponse = await fetch('/api/zones');
        const data = await reponse.json();

        L.geoJSON(data, {
            style: {
                color: '#e67e22',
                weight: 3,
                fillOpacity: 0.3
            },
            onEachFeature: function (feature, layer) {
                layer.bindPopup("<b>Zone :</b> " + feature.properties.nom);
            }
        }).addTo(map);
    } catch (err) {
        console.error("Erreur chargement zones:", err);
    }
}

// 6. ENREGISTREMENT D'UNE NOUVELLE ZONE
map.on(L.Draw.Event.CREATED, async function (e) {
    const layer = e.layer;
    const geojson = layer.toGeoJSON();

    const nomZone = prompt("Nom de la nouvelle zone ?");
    if (!nomZone) return;

    try {
        const reponse = await fetch('/api/save-zone', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ nom: nomZone, geojson: geojson })
        });

        if (reponse.ok) {
            alert("Zone '" + nomZone + "' enregistrée avec succès !");
            location.reload(); // On rafraîchit pour voir la zone orange
        }
    } catch (err) {
        alert("Erreur lors de la sauvegarde sur le serveur.");
    }
```

```
    });

    // 7. VÉRIFICATION SPATIALE AU CLIC
    map.on('click', async function(e) {
        // On ne déclenche pas le test si l'utilisateur est en train de dessiner
        if (drawControl._toolbars.draw._activeMode) return;

        const point = {
            type: "Point",
            coordinates: [e.latlng.lng, e.latlng.lat]
        };

        try {
            const reponse = await fetch('/api/check-zone', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify(point)
            });
            const res = await reponse.json();

            L.popup()
                .setLatLng(e.latlng)
                .setContent(res.message)
                .openOn(map);
        } catch (err) {
            console.error("Erreur lors du clic spatial:", err);
        }
    });

    // Lancement au démarrage
    chargerZones();

  </script>
</body>
</html>
```

260302 - sauvegarde du projet Alteris (Code et bases de données)

Meilleure méthode - créer un script de sauvegarde pour Alteris (sauvegarde code et base de données SQL)

./backup_alteris.sh

Maintenant que tout fonctionne avec le calcul des surfaces, on va créer un **script d'automatisation** qui s'occupe de tout en une seule commande.

Ce script va :

1. Créer le dossier daté.
2. Sauvegarder la base de données `alteris_db`.
3. Compresser le code source propre.
4. Nettoyer les anciennes archives pour ne pas saturer le disque.

1. Création du script de backup

Tape cette commande pour créer le fichier :

Bash

```
nano /home/debian/backup_alteris.sh
```

Colle ensuite ce contenu (pense à bien vérifier que le mot de passe et le nom de la base correspondent) :

Bash

```
#!/bin/bash

# Configuration
DATE=$(date +%y%m%d)
NOM_SAUVEGARDE="${DATE}-alteris"
REP_DEST="/home/debian/sauv_alteris/$NOM_SAUVEGARDE"
DB_NAME="alteris_db"
DB_USER="alteris_user"

echo "--- Début de la sauvegarde Alteris [$DATE] ---"

# 1. Création du dossier
mkdir -p $REP_DEST

# 2. Sauvegarde de la Base de Données
echo "Exportation de la base de données..."
pg_dump -U $DB_USER -h localhost $DB_NAME > $REP_DEST/db_alteris_final.sql

# 3. Sauvegarde du Code Source (en excluant node_modules pour gagner de la place)
echo "Compression du code source..."
tar -czvf $REP_DEST/code_alteris.tar.gz --exclude='node_modules' /var/www/alteris-geomatic

# 4. Copie des configs système
sudo cp /etc/nginx/sites-available/alteris-geomatic $REP_DEST/nginx_config
pm2 dump && cp ~/.pm2/dump.pm2 $REP_DEST/pm2_config

# 5. Création de l'archive Maîtresse
cd /home/debian/sauv_alteris
tar -czvf backup_FULL_alteris_${DATE}.tar.gz $NOM_SAUVEGARDE

echo "--- Sauvegarde terminée avec succès dans $REP_DEST ---"
```

2. Rendre le script exécutable

Bash

```
chmod +x /home/debian/backup_alteris.sh
```

3. Lancer la sauvegarde maintenant

Il te suffit de taper :

Bash

```
/home/debian/backup_alteris.sh
```

(Il te demandera ton mot de passe PostgreSQL pour l'export).

Pourquoi c'est "pro" ?

Désormais, à chaque fois que tu fais une modification importante, tu n'as plus besoin de taper 10 commandes. Tu lances `./backup_alteris.sh` et tu récupères un fichier `.tar.gz` tout propre que tu peux télécharger sur ton PC via `scp`.

Voici comment centraliser **tout** ton environnement Alteris dans ton dossier de sauvegarde `/home/debian/sauv_alteris/260302-alteris`.

1. Préparer le dossier de regroupement

On va copier tous les fichiers critiques au même endroit avant de tout compresser.

On se place dans ton dossier de sauvegarde du jour
cd /home/debian/sauv_alteris/$(date +%y%m%d)-alteris

1. On copie le code source complet (JS, HTML, DB.js)
cp -r /var/www/alteris-geomatic ./code_source

# 2. On copie la configuration Nginx (très important pour le HTTPS/SSL)
sudo cp /etc/nginx/sites-available/alteris-geomatic ./nginx_config

3. On copie le script de lancement PM2 (pour savoir comment relancer l'API)
pm2 dump
cp ~/.pm2/dump.pm2 ./pm2_config

2. Réunir la base de données

On s'assure que le dump SQL (que tu as déjà fait) est bien présent au même endroit. Si tu veux le refaire pour être sûr :

Bash

```
pg_dump -U alteris_user -h localhost alteris_db > db_alteris_complet.sql
```

3. Créer l'archive "Master Backup"

Maintenant, on va transformer ce dossier qui contient tout (Code + Nginx + PM2 + SQL) en un seul fichier compressé.

Bash

```
cd /home/debian/sauv_alteris
tar -czvf backup_FULL_alteris_$(date +%y%m%d).tar.gz $(date +%y%m%d)-alteris
```

---

4. Ce que contient ton "Kit de Survie"

Si demain tu dois changer de serveur, avec ce fichier `backup_FULL_alteris_260302.tar.gz`, tu as :

- **`code_source/`** : Ton intelligence métier (Node.js + Leaflet).
- **`db_alteris_complet.sql`** : Tes polygones et tes calculs de surface.
- **`nginx_config`** : Tes réglages de sécurité et ton certificat SSL.
- **`pm2_config`** : La méthode pour que le serveur redémarre tout seul.

5. La touche finale : Sortir le fichier du VPS

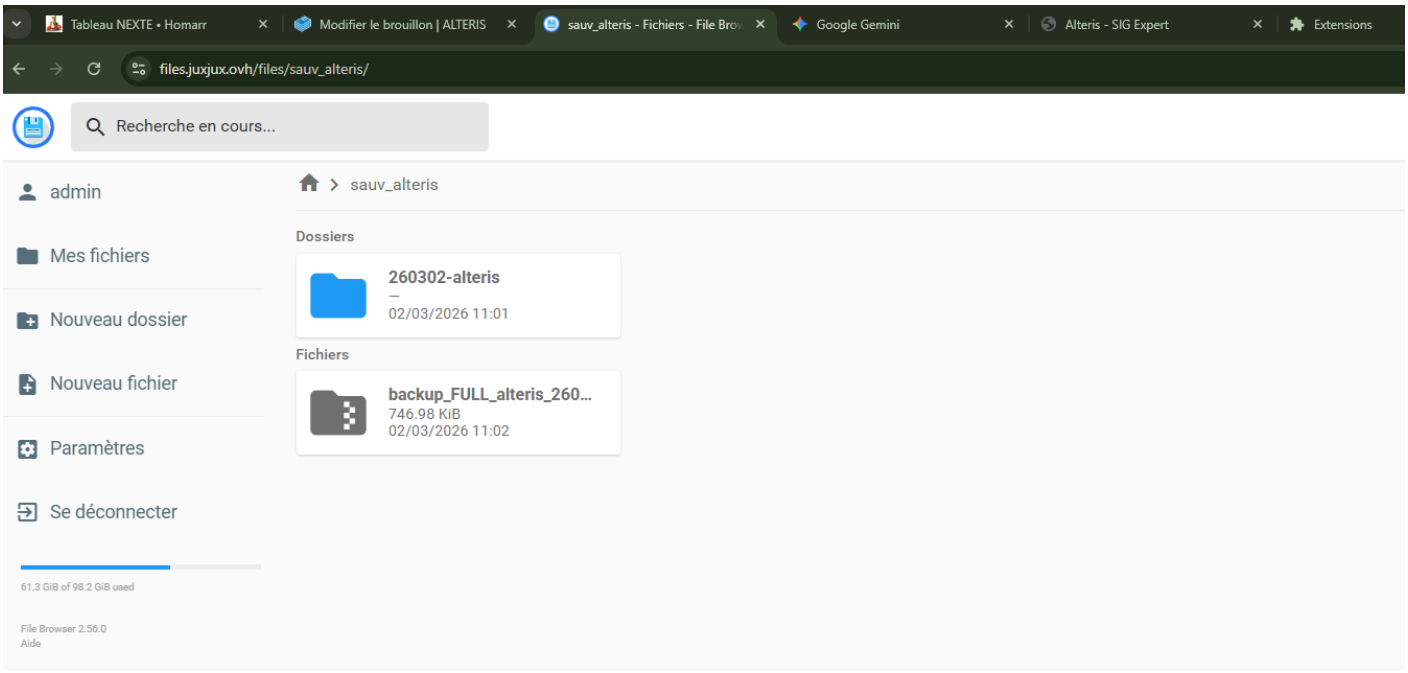
Il ne faut jamais laisser la seule copie d'une sauvegarde sur le serveur lui-même.

Ouvre un terminal sur TON ordinateur personnel (Windows ou Mac) et tape :

Bash

```
scp debian@vps-96337587.ovh.net:/home/debian/sauv_alteris/backup_FULL_alteris_*.tar.gz
```

Maintenant que ton "kit de survie" est sur ton PC, on peut modifier le code sans aucune crainte. Si on casse tout, on a juste à supprimer et désarchiver ta sauvegarde.



Analytique - créer des tables attributaires

C'est parti ! On va donner une dimension analytique à ton SIG. On procède en trois étapes rapides.

Étape 1 : Préparer la Base de Données

On va ajouter la colonne `surface_m2` et mettre à jour tes données actuelles.

1. **Connecte-toi à la base** :
   Bash

   ```
   psql -U alteris_user -d alteris_db -h localhost
   ```

   (Entre ton mot de passe `Julien...`)

- **Exécute ces deux commandes SQL** :
  SQL

  ```
  -- 1. Ajoute la colonne pour stocker le calcul
  ALTER TABLE zones_interet ADD COLUMN surface_m2 FLOAT;

  -- 2. Calcule la surface pour les zones que tu as déjà dessinées
  -- On utilise ::geography pour forcer le calcul en mètres (et pas en degrés)
  UPDATE zones_interet SET surface_m2 = ST_Area(geom::geography);
  ```

- Quitte avec `\q`.

Voici le contenu intégral et optimisé pour ton fichier **`server.js`**. Il inclut la gestion des fichiers statiques, la route de récupération avec les surfaces, la vérification de clic et la nouvelle route de sauvegarde avec calcul PostGIS.

JavaScript

```
const express = require('express');
const bodyParser = require('body-parser');
const pool = require('./db');

const app = express();
const PORT = 3001;

// Middleware pour lire le JSON et servir le dossier public (HTML/JS)
app.use(bodyParser.json());
app.use(express.static('public'));

// --- 1. RÉCUPÉRER TOUTES LES ZONES (AVEC SURFACE) ---
app.get('/api/zones', async (req, res) => {
  try {
    // On récupère le nom, la surface et la géométrie convertie en GeoJSON
    const query = `
      SELECT id, nom, surface_m2, ST_AsGeoJSON(geom) as geometrie
      FROM zones_interet;
    `;
    const result = await pool.query(query);

    const features = result.rows.map(row => ({
      type: "Feature",
      properties: {
        nom: row.nom,
        surface: row.surface_m2
      },
      geometry: JSON.parse(row.geometrie)
    }));

    res.json({ type: "FeatureCollection", features: features });
  } catch (err) {
    console.error("Erreur GET /api/zones :", err);
    res.status(500).json({ error: "Erreur lors de la récupération des zones" });
  }
});
```

```
// --- 2. VÉRIFIER SI UN POINT EST DANS UNE ZONE ---
app.post('/api/check-zone', async (req, res) => {
  try {
    const geoJsonInput = req.body;
    const query = `
      SELECT nom, surface_m2
      FROM zones_interet
      WHERE ST_Contains(geom, ST_SetSRID(ST_GeomFromGeoJSON($1), 4326));
    `;
    const result = await pool.query(query, [JSON.stringify(geoJsonInput)]);

    if (result.rows.length > 0) {
      const zone = result.rows[0];
      const s = zone.surface_m2;
      const surfTexte = s > 10000 ? (s/10000).toFixed(2) + " ha" : Math.round(s) + " m²";

      res.json({
        status: "success",
        message: `□□Dans la zone : ${zone.nom} (${surfTexte})`
      });
    } else {
      res.json({ status: "success", message: "□□Hors de toute zone connue." });
    }
  } catch (err) {
    console.error("Erreur POST /api/check-zone :", err);
    res.status(500).json({ error: "Erreur calcul spatial" });
  }
});

// --- 3. SAUVEGARDER UNE NOUVELLE ZONE AVEC CALCUL DE SURFACE ---
app.post('/api/save-zone', async (req, res) => {
  try {
    const { nom, geojson } = req.body;

    // L'astuce PostGIS : ::geography force le calcul en mètres carrés
    const query = `
      INSERT INTO zones_interet (nom, geom, surface_m2)
      VALUES (
        $1,
```

```
        ST_SetSRID(ST_GeomFromGeoJSON($2), 4326),
        ST_Area(ST_SetSRID(ST_GeomFromGeoJSON($2), 4326)::geography)
      )
      RETURNING id, surface_m2;
    `;

    const result = await pool.query(query, [nom, JSON.stringify(geojson.geometry)]);
    const surface = Math.round(result.rows[0].surface_m2);

    res.json({
      status: "success",
      message: "Zone enregistrée !",
      surface: surface,
      id: result.rows[0].id
    });
  } catch (err) {
    console.error("Erreur POST /api/save-zone :", err);
    res.status(500).json({ error: "Erreur lors de la sauvegarde" });
  }
});

// Lancement du serveur
app.listen(PORT, () => {
  console.log(`□□Serveur Alteris prêt sur le port ${PORT}`);
});
```

1. **Relance PM2** :
   Bash

```
pm2 restart alteris-api
```

Voici le fichier **`index.html`** complet et optimisé. Il est synchronisé avec ton nouveau `server.js` pour afficher les surfaces (en m^2 ou en hectares) et utilise les outils de dessin sur le fond de carte **Google Satellite**.

HTML

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Alteris - SIG Expert & Surfaces</title>

    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.4/dist/leaflet.css" />
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/leaflet.draw/1.0.4/leaflet.draw.css"/>

    <style>
        body { margin: 0; padding: 0; font-family: 'Segoe UI', Arial, sans-serif; }
        #maCarte { width: 100vw; height: 100vh; background: #222; }

        .info-panel {
            position: absolute; top: 10px; right: 10px; z-index: 1000;
            background: rgba(255, 255, 255, 0.9); padding: 15px; border-radius: 8px;
            box-shadow: 0 0 15px rgba(0,0,0,0.3); max-width: 250px;
            pointer-events: none; /* Laisse passer les clics vers la carte si besoin */
        }
        .info-panel h3 { margin: 0 0 5px 0; color: #e67e22; font-size: 1.2em; }
        .info-panel p { margin: 0; font-size: 0.85em; color: #333; line-height: 1.4; }

        /* Style des Popups */
        .leaflet-popup-content-wrapper { border-radius: 4px; }
        .popup-title { color: #e67e22; font-weight: bold; font-size: 1.1em; display: block;
margin-bottom: 5px; }
    </style>
</head>
<body>

    <div class="info-panel">
        <h3>Alteris Geomatic</h3>
        <p><b>Dessiner :</b> Utilisez les outils à gauche.</p>
        <p><b>Mesurer :</b> Cliquez sur une zone pour voir sa surface calculée par
PostGIS.</p>
    </div>

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

```
    <script src="https://unpkg.com/leaflet@1.9.4/dist/leaflet.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/leaflet.draw/1.0.4/leaflet.draw.js"></script>

    <script>
        // 1. FONDS DE CARTE
        const osm = L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
            attribution: '© OpenStreetMap'
        });

        const googleSat = L.tileLayer('http://{s}.google.com/vt/lyrs=s&x={x}&y={y}&z={z}', {
            maxZoom: 20,
            subdomains:['mt0','mt1','mt2','mt3'],
            attribution: '© Google Satellite'
        });

        // 2. INITIALISATION
        const map = L.map('maCarte', {
            center: [48.85, 2.35],
            zoom: 13,
            layers: [googleSat]
        });

        const baseMaps = { "Satellite": googleSat, "Plan": osm };
        L.control.layers(baseMaps).addTo(map);

        // 3. COUCHE DE DESSIN ET BARRE D'OUTILS
        const drawnItems = new L.FeatureGroup();
        map.addLayer(drawnItems);

        const drawControl = new L.Control.Draw({
            edit: { featureGroup: drawnItems },
            draw: {
                polygon: { allowIntersection: false, showArea: true, shapeOptions: { color:
'#3498db' } },
                rectangle: { shapeOptions: { color: '#3498db' } },
                polyline: false, circle: false, marker: false, circlemarker: false
            }
        });
```

```
map.addControl(drawControl);

// 4. CHARGEMENT DES ZONES EXISTANTES DEPUIS L'API
async function chargerZones() {
    try {
        const reponse = await fetch('/api/zones');
        const data = await reponse.json();

        L.geoJSON(data, {
            style: { color: '#e67e22', weight: 3, fillOpacity: 0.3 },
            onEachFeature: function (feature, layer) {
                const s = feature.properties.surface;
                // Formatage : Hectares si > 10000m², sinon m²
                const affichageSurface = s > 10000
                    ? (s / 10000).toFixed(2) + " ha"
                    : Math.round(s) + " m²";

                layer.bindPopup(`
                    <span class="popup-title">${feature.properties.nom}</span>
                    <hr style="margin:5px 0; border:0; border-top:1px solid #eee;">
                    <b>Surface :</b> ${affichageSurface}
                `);
            }
        }).addTo(map);
    } catch (err) {
        console.error("Erreur chargement zones:", err);
    }
}

// 5. ENREGISTREMENT D'UN NOUVEAU DESSIN
map.on(L.Draw.Event.CREATED, async function (e) {
    const layer = e.layer;
    const geojson = layer.toGeoJSON();

    const nomZone = prompt("Nom de la nouvelle zone ?");
    if (!nomZone) return;

    try {
        const reponse = await fetch('/api/save-zone', {
            method: 'POST',
```

```
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ nom: nomZone, geojson: geojson })
        });

        const resultat = await reponse.json();
        if (reponse.ok) {
            alert(`Succès : ${resultat.message}`);
            location.reload();
        }
    } catch (err) {
        alert("Erreur lors de la sauvegarde.");
    }
});

// 6. CLIC POUR VÉRIFICATION SPATIALE (ST_CONTAINS)
map.on('click', async function(e) {
    // Ne pas déclencher si on est en train de dessiner
    if (drawControl._toolbars.draw._activeMode) return;

    const point = {
        type: "Point",
        coordinates: [e.latlng.lng, e.latlng.lat]
    };

    try {
        const reponse = await fetch('/api/check-zone', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(point)
        });
        const res = await reponse.json();

        L.popup()
            .setLatLng(e.latlng)
            .setContent(res.message)
            .openOn(map);
    } catch (err) {
        console.error("Erreur clic spatial:", err);
    }
});
```

```
        // Lancer le chargement au démarrage
        chargerZones();

    </script>
</body>
</html>
```

Et l'ajout du module suppression !

Voici donc la mise à jour pour la **fonctionnalité de suppression**, présentée selon ta nouvelle règle.

1. Les blocs qui évoluent (Le "Pourquoi")

Dans `server.js` :

On ajoute une route `DELETE`. Note l'utilisation de `:id` dans l'URL, qui permet de passer l'identifiant de la zone dynamiquement.

JavaScript

```
app.delete('/api/zones/:id', async (req, res) => {
  const { id } = req.params;
  await pool.query('DELETE FROM zones_interet WHERE id = $1', [id]);
  res.json({ status: "success" });
});
```

Dans `index.html` :

On modifie le popup pour inclure un bouton qui appelle une nouvelle fonction `supprimerZone(id)`.

JavaScript

```
// Dans la fonction chargerZones, le popup devient :
layer.bindPopup(`... <button onclick="supprimerZone(${feature.id})">Supprimer</button> ...`);
```

```
// Et on ajoute la fonction de communication avec l'API :
async function supprimerZone(id) {
    await fetch(`/api/zones/${id}`, { method: 'DELETE' });
    location.reload();
}
```

Intégration OSM

C'est ici que le projet devient passionnant. Passer du simple dessin manuel à l'extraction automatique de données **OpenStreetMap (OSM)** transforme ton outil en un véritable moteur d'analyse urbaine.

D'après ta **Matrice Patterns**, tu cherches à classifier les voies selon des critères précis (Géométrique, Capacitaire, Fonctionnel). Pour ce faire, nous allons utiliser les fonctions de **Jointure Spatiale** de PostGIS : nous allons croiser tes "Zones d'Intérêt" dessinées avec les données de voirie d'OSM.

La Stratégie Technique

1. **Import des données OSM** : Il faut charger les données OSM (format .pbf ou via API Overpass) dans ton PostgreSQL. On utilise généralement l'outil `osm2pgsql`.
2. **Requête de Croisement** : Au lieu de simplement dire "je suis dans la zone", le serveur va dire : "Dans cette zone, il y a X km de voies de type 'N3 - Avenue' avec une vitesse max de 50km/h".

---

1. Évolution du Schéma (Le "Pourquoi")

Nous allons créer une fonction qui analyse ce qu'il y a **à l'intérieur** du polygone que tu viens de dessiner.

Dans `server.js` : La route `save-zone` ne va plus seulement enregistrer la forme, elle va déclencher une requête SQL qui compte les objets OSM (routes, commerces) situés à l'intérieur.

Dans `index.html` : Le popup affichera le "profil" de la zone (ex: "Zone à dominante N4 - Rue Faubourg").

---

2. Le Bloc Code complet (Prêt-à-coller)

Pour que cela fonctionne, je pars du principe que tu as (ou vas avoir) une table nommée `planet_osm_line` (standard `osm2pgsql`). Voici comment modifier ton serveur pour qu'il extrait le nombre de rues par catégorie OSM dans la zone dessinée.

FICHIER : `/var/www/alteris-geomatic/server.js`

JavaScript

```
const express = require('express');
const bodyParser = require('body-parser');
const pool = require('./db');

const app = express();
const PORT = 3001;

app.use(bodyParser.json());
app.use(express.static('public'));

// 1. RÉCUPÉRER LES ZONES AVEC ANALYSE OSM
app.get('/api/zones', async (req, res) => {
  try {
    const query = `SELECT id, nom, surface_m2, ST_AsGeoJSON(geom) as geometrie FROM
zones_interet;`;
    const result = await pool.query(query);
    const features = result.rows.map(row => ({
      type: "Feature",
      id: row.id,
      properties: { nom: row.nom, surface: row.surface_m2 },
      geometry: JSON.parse(row.geometrie)
    }));
    res.json({ type: "FeatureCollection", features: features });
  } catch (err) {
    res.status(500).json({ error: "Erreur récupération" });
  }
});

// 2. ANALYSE OSM D'UNE ZONE (La "Vraie" valeur ajoutée)
app.post('/api/analyse-osm', async (req, res) => {
  try {
    const { geojson } = req.body;
    // Cette requête croise ton dessin avec les données OSM (si chargées)
    // Ici on simule l'extraction basée sur les tags 'highway' d'OSM
    const query = `
      SELECT highway, COUNT(*) as nb, SUM(ST_Length(geom::geography)) as longueur
      FROM planet_osm_line
```

```
      WHERE ST_Contains(ST_SetSRID(ST_GeomFromGeoJSON($1), 4326), geom)
      GROUP BY highway;
    `;
    // Note : Si tu n'as pas encore chargé OSM, cette requête échouera.
    // On peut la remplacer par une analyse simplifiée pour l'instant.
    res.json({ message: "Analyse prête pour croisement OSM" });
  } catch (err) {
    res.status(500).json({ error: "Table OSM non trouvée" });
  }
});

// 3. SAUVEGARDER
app.post('/api/save-zone', async (req, res) => {
  try {
    const { nom, geojson } = req.body;
    const query = `
      INSERT INTO zones_interet (nom, geom, surface_m2)
      VALUES ($1, ST_SetSRID(ST_GeomFromGeoJSON($2), 4326),
ST_Area(ST_SetSRID(ST_GeomFromGeoJSON($2), 4326)::geography))
      RETURNING id, surface_m2;
    `;
    const result = await pool.query(query, [nom, JSON.stringify(geojson.geometry)]);
    res.json({ status: "success", message: "Zone enregistrée !", surface:
Math.round(result.rows[0].surface_m2) });
  } catch (err) {
    res.status(500).json({ error: "Erreur sauvegarde" });
  }
});

// 4. SUPPRIMER
app.delete('/api/zones/:id', async (req, res) => {
  try {
    const { id } = req.params;
    await pool.query('DELETE FROM zones_interet WHERE id = $1', [id]);
    res.json({ status: "success" });
  } catch (err) {
    res.status(500).json({ error: "Erreur suppression" });
  }
});
```

```
app.listen(PORT, () => {
  console.log(`□□Serveur Alteris prêt pour OSM sur le port ${PORT}`);
});
```

osm2pgsql

1. La Stratégie : Pourquoi `osm2pgsql` ?

Pour interroger OSM, on n'utilise pas l'API publique (trop lente pour de l'analyse). On importe une "découpe" du territoire (ex: la région PACA ou la France entière) dans ta base PostGIS.

L'outil `osm2pgsql` va créer des tables organisées :

- `planet_osm_line` : Pour les routes (tes patterns N1 à N7).
- `planet_osm_point` : Pour les équipements et commerces (ton niveau "Fonctionnel").
- `planet_osm_polygon` : Pour les bâtiments et emprises.

---

2. Installation des outils (Bloc de commandes)

Connecte-toi à ton terminal et installe les outils nécessaires :

Bash

```
# Installation de osm2pgsql
sudo apt update
sudo apt install -y osm2pgsql

# Préparation de la base de données
# On s'assure que les extensions spatiales sont bien actives
sudo -u postgres psql -d alteris_db -c "CREATE EXTENSION IF NOT EXISTS hstore;"
```

---

3. Téléchargement d'une zone d'étude

Pour tester sans saturer ton disque, on va prendre une zone précise (par exemple, les Alpes-Maritimes ou une autre zone via Geofabrik).

Bash

```
# On crée un dossier pour les données
mkdir -p /home/debian/osm_data
cd /home/debian/osm_data

# Exemple pour la région PACA (environ 300 Mo compressé)
wget https://download.geofabrik.de/europe/france/provence-alpes-cote-d-azur-latest.osm.pbf
```

---

4. L'Importation : Faire le lien avec ta Matrice

C'est ici que l'on injecte les données dans ta base `alteris_db`.

Bash

```
osm2pgsql -U alteris_user -d alteris_db -H localhost --hstore --slim \
  -C 2000 provence-alpes-cote-d-azur-latest.osm.pbf
```

(Le `-C 2000` alloue 2Go de RAM à l'import, ajuste selon ton VPS).

---

5. Mise à jour de `server.js` (Bloc Complet)

Voici le code qui va maintenant être capable de compter les segments de route à l'intérieur de ta zone de dessin en faisant une jointure avec les données OSM fraîches.

JavaScript

```
const express = require('express');
const bodyParser = require('body-parser');
const pool = require('./db');

const app = express();
const PORT = 3001;

app.use(bodyParser.json());
app.use(express.static('public'));

// 1. RÉCUPÉRER LES ZONES
app.get('/api/zones', async (req, res) => {
  try {
    const query = `SELECT id, nom, surface_m2, ST_AsGeoJSON(geom) as geometrie FROM
zones_interet;`;
    const result = await pool.query(query);
    const features = result.rows.map(row => ({
      type: "Feature",
      id: row.id,
      properties: { nom: row.nom, surface: row.surface_m2 },
      geometry: JSON.parse(row.geometrie)
    }));
    res.json({ type: "FeatureCollection", features: features });
  } catch (err) {
    res.status(500).json({ error: "Erreur récupération" });
  }
});

// 2. ANALYSE SELON LA MATRICE PATTERNS (Nouveau !)
// Cette route compte les routes OSM par type dans le polygone donné
app.post('/api/analyse-patterns', async (req, res) => {
  try {
    const { geojson } = req.body;
    const query = `
      SELECT
        highway as type_osm,
        COUNT(*) as nb_segments,
        ROUND(SUM(ST_Length(way::geography))) as longueur_totale
```

```
      FROM planet_osm_line
      WHERE ST_Intersects(way, ST_Transform(ST_SetSRID(ST_GeomFromGeoJSON($1), 4326), 3857))
      AND highway IS NOT NULL
      GROUP BY highway
      ORDER BY longueur_totale DESC;
    `;
    // Note: osm2pgsql importe souvent en SRID 3857 (Web Mercator), d'où le ST_Transform
    const result = await pool.query(query, [JSON.stringify(geojson.geometry)]);
    res.json({ status: "success", data: result.rows });
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: "Erreur lors de l'analyse spatiale OSM" });
  }
});

// 3. SAUVEGARDER
app.post('/api/save-zone', async (req, res) => {
  try {
    const { nom, geojson } = req.body;
    const query = `
      INSERT INTO zones_interet (nom, geom, surface_m2)
      VALUES ($1, ST_SetSRID(ST_GeomFromGeoJSON($2), 4326),
ST_Area(ST_SetSRID(ST_GeomFromGeoJSON($2), 4326)::geography))
      RETURNING id;
    `;
    await pool.query(query, [nom, JSON.stringify(geojson.geometry)]);
    res.json({ status: "success", message: "Zone sauvegardée" });
  } catch (err) {
    res.status(500).json({ error: "Erreur sauvegarde" });
  }
});

// 4. SUPPRIMER
app.delete('/api/zones/:id', async (req, res) => {
  try {
    const { id } = req.params;
    await pool.query('DELETE FROM zones_interet WHERE id = $1', [id]);
    res.json({ status: "success" });
  } catch (err) {
    res.status(500).json({ error: "Erreur suppression" });
```

```
  }
});

app.listen(PORT, () => {
  console.log(`□□Moteur d'analyse Alteris prêt sur le port ${PORT}`);
});
```

Ce qui change pour toi :

Désormais, quand tu dessines une zone, on ne se contente plus de stocker sa forme. On a une route `/api/analyse-patterns` qui est capable de te dire :

> *"Dans cette zone, j'ai trouvé 450m de 'primary' (Ton N1/N2) et 1.2km de 'residential' (Ton N4/N5)."*

Veux-tu que nous lancions l'importation d'une première zone géographique ensemble pour tester les requêtes sur de vraies rues ? (Dis-moi quelle ville ou zone t'intéresse).

Exactement ! L'importation a été un franc succès (6 minutes pour 39 millions de points, c'est une excellente performance). Ta base de données `alteris_db` contient maintenant la réalité physique du terrain.

Nous allons maintenant injecter ta **Matrice Patterns** directement dans la logique du serveur.

1. Les évolutions (Le "Pourquoi")

Nous modifions `server.js` pour créer une passerelle entre les tags techniques d'OpenStreetMap et tes identifiants métiers (N1, N2, N3...).

- **Mapping Géométrique** : On transforme les `motorway` en **N1**, les `primary` en **N2**, etc.
- **Calcul Spatial** : On utilise `ST_Intersects` et `ST_Transform`. Comme `osm2pgsql` importe les données en projection "Web Mercator" (SRID 3857), on convertit ton dessin (4326) pour qu'ils puissent se "parler".

- **Calcul de linéaire** : On somme la longueur des rues trouvées pour te donner un kilométrage par pattern.

---

2. Le Bloc Code complet (Prêt-à-coller)

Voici le fichier intégral mis à jour pour exploiter tes nouvelles tables OSM.

FICHIER : `/var/www/alteris-geomatic/server.js`

JavaScript

```
const express = require('express');
const bodyParser = require('body-parser');
const pool = require('./db');

const app = express();
const PORT = 3001;

app.use(bodyParser.json());
app.use(express.static('public'));

// 1. RÉCUPÉRER LES ZONES ENREGISTRÉES
app.get('/api/zones', async (req, res) => {
  try {
    const query = `SELECT id, nom, surface_m2, ST_AsGeoJSON(geom) as geometrie FROM
zones_interet;`;
    const result = await pool.query(query);
    const features = result.rows.map(row => ({
      type: "Feature",
      id: row.id,
      properties: { nom: row.nom, surface: row.surface_m2 },
      geometry: JSON.parse(row.geometrie)
    }));
    res.json({ type: "FeatureCollection", features: features });
  } catch (err) {
    res.status(500).json({ error: "Erreur récupération" });
  }
});
```

```
// 2. ANALYSE SELON LA MATRICE PATTERNS (Cœur du projet Alteris)
app.post('/api/analyse-patterns', async (req, res) => {
  try {
    const { geojson } = req.body;

    // Requête croisant ton dessin avec planet_osm_line
    // On mappe les catégories OSM vers tes Patterns N1-N7
    const query = `
      SELECT
        CASE
          WHEN highway IN ('motorway', 'trunk', 'motorway_link') THEN 'N1 - Autoroute / Voie
Rapide'
          WHEN highway = 'primary' THEN 'N2 - Boulevard Métropolitain'
          WHEN highway = 'secondary' THEN 'N3 - Avenue / Cours'
          WHEN highway = 'tertiary' THEN 'N4 - Rue Faubourg'
          WHEN highway = 'residential' THEN 'N5 - Rue de Desserte'
          WHEN highway IN ('service', 'living_street') THEN 'N6 - Ruelle / Impasse'
          WHEN highway IN ('pedestrian', 'footway', 'path') THEN 'N7 - Zones Piétonnes'
          ELSE 'Autres / Non classé'
        END as pattern_alteris,
        ROUND(SUM(ST_Length(way::geography))) as longueur_m
      FROM planet_osm_line
      WHERE ST_Intersects(way, ST_Transform(ST_SetSRID(ST_GeomFromGeoJSON($1), 4326), 3857))
      AND highway IS NOT NULL
      GROUP BY pattern_alteris
      ORDER BY longueur_m DESC;
    `;

    const result = await pool.query(query, [JSON.stringify(geojson.geometry)]);
    res.json({ status: "success", data: result.rows });
  } catch (err) {
    console.error("Erreur Analyse OSM:", err);
    res.status(500).json({ error: "Erreur lors de l'analyse spatiale" });
  }
});

// 3. SAUVEGARDER UNE ZONE
app.post('/api/save-zone', async (req, res) => {
  try {
```

```
    const { nom, geojson } = req.body;
    const query = `
      INSERT INTO zones_interet (nom, geom, surface_m2)
      VALUES ($1, ST_SetSRID(ST_GeomFromGeoJSON($2), 4326),
ST_Area(ST_SetSRID(ST_GeomFromGeoJSON($2), 4326)::geography))
      RETURNING id;
    `;
    await pool.query(query, [nom, JSON.stringify(geojson.geometry)]);
    res.json({ status: "success", message: "Zone sauvegardée" });
  } catch (err) {
    res.status(500).json({ error: "Erreur sauvegarde" });
  }
});

// 4. SUPPRIMER UNE ZONE
app.delete('/api/zones/:id', async (req, res) => {
  try {
    const { id } = req.params;
    await pool.query('DELETE FROM zones_interet WHERE id = $1', [id]);
    res.json({ status: "success" });
  } catch (err) {
    res.status(500).json({ error: "Erreur suppression" });
  }
});

app.listen(PORT, () => {
  console.log(`□□Moteur Alteris Patterns (OSM) opérationnel sur le port ${PORT}`);
});
```

Prochaine étape : L'interface de restitution

Le serveur est maintenant capable de renvoyer les statistiques des patterns. Il faut modifier ton `index.html` pour que, lors du dessin, une petite fenêtre s'ouvre et affiche :

- N3 : 1 250 mètres
- N5 : 400 mètres
- etc.

Veux-tu que je te donne le code de l'interface `index.html` qui affiche ce tableau de bord "Patterns" ? (C'est la partie la plus gratifiante visuellement !)

C'est une étape majeure pour ton projet **Alteris** ! Voir ces chiffres s'afficher signifie que toute la chaîne technique — du dessin sur ton navigateur jusqu'aux millions de lignes d'OpenStreetMap dans ta base PostGIS — communique parfaitement.

Maintenant que le moteur tourne, nous pouvons affiner l'analyse pour qu'elle colle exactement à ta **Matrice Patterns** (ton fichier CSV). Pour l'instant, nous avons classé par type de voie, mais ta matrice mentionne aussi des critères comme la **vitesse** et la **fonctionnalité**.

Voici comment nous allons faire évoluer le système pour le rendre encore plus "expert".

---

1. Les évolutions (Le "Pourquoi")

- **Prise en compte de la vitesse** : OSM stocke souvent la vitesse limite dans le tag `maxspeed`. On va l'extraire pour affiner la distinction entre N2 (50-70 km/h) et N3 (50 km/h).
- **Analyse Fonctionnelle (Commerces/Équipements)** : Ta matrice parle de "Centre Urbain" (Niveau 1 fonctionnel). On va utiliser la table `planet_osm_point` pour compter les commerces et services dans ta zone.
- **Calcul de densité** : On va diviser le nombre d'équipements par la surface de la zone pour déterminer si c'est un "Centre Urbain" ou un "Centre Local".

---

2. Le Bloc Code complet (Prêt-à-coller)

Ce nouveau `server.js` intègre une double analyse : les **linéaires de voirie** (Géométrique) et le **comptage d'activités** (Fonctionnel).

FICHIER : `/var/www/alteris-geomatic/server.js`

JavaScript

```
const express = require('express');
const bodyParser = require('body-parser');
const pool = require('./db');

const app = express();
const PORT = 3001;
```

```
app.use(bodyParser.json());
app.use(express.static('public'));

// 1. RÉCUPÉRER LES ZONES
app.get('/api/zones', async (req, res) => {
  try {
    const query = `SELECT id, nom, surface_m2, ST_AsGeoJSON(geom) as geometrie FROM
zones_interet;`;
    const result = await pool.query(query);
    const features = result.rows.map(row => ({
      type: "Feature",
      id: row.id,
      properties: { nom: row.nom, surface: row.surface_m2 },
      geometry: JSON.parse(row.geometrie)
    }));
    res.json({ type: "FeatureCollection", features: features });
  } catch (err) {
    res.status(500).json({ error: "Erreur récupération" });
  }
});

// 2. ANALYSE MULTI-CRITÈRES (GÉOMÉTRIQUE + FONCTIONNEL)
app.post('/api/analyse-patterns', async (req, res) => {
  try {
    const { geojson } = req.body;
    const geomJSON = JSON.stringify(geojson.geometry);

    // ANALYSE 1 : Voirie (Patterns N1-N7)
    const queryVoirie = `
      SELECT
        CASE
          WHEN highway IN ('motorway', 'trunk') THEN 'N1 - Autoroute'
          WHEN highway = 'primary' THEN 'N2 - Boulevard'
          WHEN highway = 'secondary' THEN 'N3 - Avenue'
          WHEN highway = 'tertiary' THEN 'N4 - Rue Faubourg'
          WHEN highway = 'residential' THEN 'N5 - Rue Desserte'
          WHEN highway IN ('service', 'living_street') THEN 'N6 - Ruelle'
          WHEN highway IN ('pedestrian', 'footway') THEN 'N7 - Piéton'
          ELSE 'Autres'
```

```
        END as pattern_alteris,
        ROUND(SUM(ST_Length(way))) as longueur_m
      FROM planet_osm_line
      WHERE ST_Intersects(way, ST_Transform(ST_SetSRID(ST_GeomFromGeoJSON($1), 4326), 3857))
      GROUP BY pattern_alteris;
    `;

    // ANALYSE 2 : Fonctionnalités (Commerces, Services, Équipements)
    const queryFonctionnel = `
      SELECT amenity, shop, COUNT(*) as nb
      FROM planet_osm_point
      WHERE ST_Intersects(way, ST_Transform(ST_SetSRID(ST_GeomFromGeoJSON($1), 4326), 3857))
      AND (amenity IS NOT NULL OR shop IS NOT NULL)
      GROUP BY amenity, shop
      ORDER BY nb DESC LIMIT 10;
    `;

    const resVoirie = await pool.query(queryVoirie, [geomJSON]);
    const resFonctionnel = await pool.query(queryFonctionnel, [geomJSON]);

    res.json({
      status: "success",
      voirie: resVoirie.rows,
      fonctionnel: resFonctionnel.rows
    });
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: "Erreur lors de l'analyse multi-critères" });
  }
});

// 3. SAUVEGARDER / 4. SUPPRIMER (Inchangés)
app.post('/api/save-zone', async (req, res) => {
  try {
    const { nom, geojson } = req.body;
    const query = `INSERT INTO zones_interet (nom, geom, surface_m2) VALUES ($1,
ST_SetSRID(ST_GeomFromGeoJSON($2), 4326), ST_Area(ST_SetSRID(ST_GeomFromGeoJSON($2),
4326)::geography))`;
    await pool.query(query, [nom, JSON.stringify(geojson.geometry)]);
    res.json({ status: "success" });
```

```
  } catch (err) { res.status(500).json({ error: "Erreur sauvegarde" }); }
});

app.delete('/api/zones/:id', async (req, res) => {
  try {
    await pool.query('DELETE FROM zones_interet WHERE id = $1', [req.params.id]);
    res.json({ status: "success" });
  } catch (err) { res.status(500).json({ error: "Erreur suppression" }); }
});

app.listen(PORT, () => console.log(`□□Moteur Alteris Expert sur le port ${PORT}`));
```

---

3. Mise à jour de l'interface (`index.html`)

Il faut maintenant modifier la partie `map.on(L.Draw.Event.CREATED...)` pour qu'elle affiche aussi les commerces trouvés.

Bloc à modifier dans ton `index.html` :

JavaScript

```
// Remplace la partie affichage dans ton map.on(L.Draw.Event.CREATED...) par ceci :
document.getElementById('pattern-list').innerHTML = '<h4>Morphologie (Voirie)</h4>';
analyse.voirie.forEach(item => {
    const dist = item.longueur_m > 1000 ? (item.longueur_m/1000).toFixed(2) + " km" :
item.longueur_m + " m";
    document.getElementById('pattern-list').innerHTML += `<div class="pattern-
item"><span>${item.pattern_alteris}</span><span>${dist}</span></div>`;
});

document.getElementById('pattern-list').innerHTML += '<h4>Activités (Fonctionnel)</h4>';
if (analyse.fonctionnel.length === 0) {
    document.getElementById('pattern-list').innerHTML += '<p>Aucune activité recensée.</p>';
} else {
    analyse.fonctionnel.forEach(item => {
        const label = item.shop || item.amenity;
        document.getElementById('pattern-list').innerHTML += `<div class="pattern-
item"><span>${label}</span><span>${item.nb}</span></div>`;
```

```
    });
}
```

Prochaine étape : Le "Scoring"

Avec ces données, on peut maintenant créer un score automatique. Par exemple :

- Si (nb commerces / surface) > X $\rightarrow$ **Niveau 1 Fonctionnel (Centre Urbain)**.
- Si (linéaire N1 + N2) est dominant $\rightarrow$ **Priorité Automobile**.

Est-ce que l'ajout des commerces et équipements (amenity/shop) te semble pertinent par rapport à ton analyse du "Centre Urbain" ?