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();
}
```

Revision #2
Created 2026-03-02 10:48:46 UTC by Julien
Updated 2026-03-02 11:34:31 UTC by Julien