Programming Logic for GIS

From Markup to Interactive Architecture

Program Sarjana Terapan Teknologi Survei dan Pemetaan Dasar
Departemen Teknologi Kebumian
Sekolah Vokasi, UGM

Ismail Sunni | Geospatial Software Engineer | Camptocamp DE

This Presentation

https://github.com/ismailsunni/web-gis-2026

About Me

- 🌍 Open source geospatial since 2012
- 🛠️ Full-stack GIS development (library, desktop, web, etc.)
- 💼 Currently @ Camptocamp DE
- 👥 Founder of **QGIS ID** and **Python Jogja**
- 🔗 ismailsunni.id | github.com/ismailsunni

Learning Objectives

By the end of this session, you should be able to:

✅ Explain the role of **JavaScript in WebGIS**
✅ Understand **DOM manipulation**
✅ Understand **event-driven systems**
✅ Explain why **OOP is important in WebGIS**
✅ Build a **simple interactive mini WebGIS**

Session Outline

1. **Architecture** — WebGIS structure & why client-side matters
2. **JavaScript Concepts** — Variables, Functions, Objects
3. **OOP & Classes** — Modeling spatial entities
4. **Event-Driven Systems** — How interaction works
5. **DOM Mastery** — Manipulating the browser
6. **Hands-on Coding** — Build a mini WebGIS from scratch
7. **Reflection & Next Steps** — Key takeaways

1 Architecture

WebGIS Structure & Client-Side Logic

WebGIS Architecture Overview

Client Side

- **HTML** — Structure
- **CSS** — Style
- **JavaScript** — Logic & Interaction
- **GIS Library** — Leaflet / OpenLayers

Server Side

- Database & Spatial API
- GeoServer

Focus: Client-Side Logic

2️⃣ JavaScript Concepts

Foundation for WebGIS Programming

The Power of JavaScript

Without JavaScript	With JavaScript
Static layout	Zoom & pan
No interaction	Click events
No dynamic updates	Add/remove layers
	Popups & feedback

JavaScript makes WebGIS alive.

JavaScript Fundamentals for GIS

Variables in Spatial Context

```
let layerName = "Roads";
let featureCount = 120;
let isVisible = true;
```

Why types matter: Coordinates must be numbers, attributes are strings, visibility is boolean.

Functions = System Behavior

```
function toggleLayer() {
    console.log("Layer toggled");
}
```

Common GIS functions:

- `zoomIn()` | `zoomOut()`
- `addLayer()` | `removeLayer()`
- `panTo()`

Objects = Spatial Entities

```
let layer = {
    name: "Roads",
    visible: false
};

let marker = {
    lat: -7.5,
    lng: 110.3
};
```

WebGIS = collection of interacting objects

3 Object-Oriented Programming (OOP)

Modeling Spatial Systems

What is OOP?

Object-Oriented Programming is a design approach that organizes code around **objects** (things) rather than just functions (actions).

Key Principles:

- **Encapsulation** — Bundle data + behavior together
- **Properties** — What an object has (state)
- **Methods** — What an object does (behavior)

OOP Examples

Real-world Analogy:

A **Car** (object) has:

- Properties: color, speed, fuel
- Methods: start(), drive(), brake()

In WebGIS:

A **Layer** object has:

- Properties: name, visibility
- Methods: toggle(), zoomTo()

Objects bundle related data and behavior together.

Why OOP Matters in GIS

Spatial systems have complexity:

- Map, Layer, Marker, Feature, Control
- Each has **properties** and **behaviors**

Problem: Managing 10+ layers without structure → chaos

Solution: Object-Oriented Design

Class as Blueprint

```
class Layer {
    constructor(name) {
        this.name = name;
        this.visible = false;
    }

    toggle() {
        this.visible = !this.visible;
    }
}

let roadLayer = new Layer("Roads");
roadLayer.toggle();
```

4 Event-Driven Systems

How WebGIS Responds to Users

Event-Driven Architecture

WebGIS reacts to user actions:

```
User Action → Event → Event Listener → Function → DOM Update
```

Examples: click, zoom, drag, hover

Event Listener Example

```
document
  .getElementById("btnLayer")
  .addEventListener("click", function() {
      toggleLayer();
  });
```

5 DOM Mastery

Manipulating the Browser

What is DOM?

Document Object Model (DOM) is a programming interface that represents HTML as a tree of **accessible objects**.

Key Concept:

- **HTML** = Static text markup
- **DOM** = Live, interactive object representation

```
<!-- HTML: just text -->
<div id="map"></div>

<!-- Every HTML element becomes a DOM object -->
<button id="toggleLayers">Toggle</button>
```

Think of DOM as the "API" to interact with HTML dynamically.

DOM in Practice

Four essential operations:

```
// 1. Access an element
document.getElementById("map");

// 2. Read/Modify content
map.innerHTML = "<h3>Layers</h3>";

// 3. Control classes
map.classList.add("active");

// 4. Change styles
map.style.display = "none";
```

These operations make WebGIS interactive!

Why DOM is Critical

Because the map lives in a `<div>`, and everything is manipulated dynamically:

- Markers are DOM elements
- Popups appear/disappear
- Layers are toggled on/off

Without DOM manipulation → no interactivity

The DOM Tree Structure

HTML becomes a **tree of accessible objects:**

```
document
  └── html
      ├── head
      │   └── title
      └── body
          ├── div#map
          ├── button#btnZoom
          └── div#legend
              └── p
```

Each element is an **object with properties and methods**

Essential DOM Methods

```
// Find elements
document.getElementById("map");
document.querySelector(".layer");
document.querySelectorAll("button");

// Modify content
element.textContent = "New text";
element.innerHTML = "<span>HTML</span>";

// Add/remove classes
element.classList.add("active");
element.classList.toggle("hidden");

// Change styles
element.style.display = "none";
element.style.backgroundColor = "blue";
```

DOM in WebGIS Context

Real WebGIS example:

```
<div id="map"></div>
<button id="toggleLayers">Toggle Layers</button>
<div id="info">Select a feature</div>
```

```
// When user clicks button → update map div → layers change
document.getElementById("toggleLayers").addEventListener("click", () => {
    document.getElementById("map").innerHTML = updateLayers();
});

// When user clicks marker → update info div
map.on("click", (feature) => {
    document.getElementById("info").textContent = feature.properties.name;
});
```

The DOM is your WebGIS interface to the browser.

6 Hands-on Coding

Building a Mini WebGIS from Scratch

🔨 Live Coding: Mini WebGIS

Build a simple interactive map **without libraries** to understand the architecture.

Step 1: HTML Structure

```
<!DOCTYPE html>
<html>
    <head>
        <title>Mini WebGIS</title>
        <style>
            #map {
                width: 100%;
                height: 400px;
                background: linear-gradient(to bottom, lightblue, lightgreen);
                position: relative;
                border: 1px solid #ccc;
            }
            .marker {
                width: 12px;
                height: 12px;
                background: red;
                position: absolute;
                border-radius: 50%;
                cursor: pointer;
            }
        </style>
    </head>
    <body>
        <h2>Mini WebGIS</h2>
        <button id="btnAdd">Add Marker</button>
        <div id="map"></div>
        <script src="app.js"></script>
    </body>
</html>
```

Step 2: JavaScript Logic

```
let map = document.getElementById("map");

document
  .getElementById("btnAdd")
  .addEventListener("click", function() {
    let marker = document.createElement("div");
    marker.className = "marker";

    marker.style.left = Math.random() * 380 + "px";
    marker.style.top = Math.random() * 380 + "px";

    map.appendChild(marker);

    marker.addEventListener("click", function() {
        alert("Marker clicked!");
    });
});
```

What We Demonstrated

✅ DOM creation
✅ Event handling
✅ Object generation
✅ Dynamic rendering

This is how GIS libraries work internally!

Step 3: OOP Refactor

```
class Marker {
    constructor(x, y) {
        this.element = document.createElement("div");
        this.element.className = "marker";
        this.element.style.left = x + "px";
        this.element.style.top = y + "px";
        this.element.addEventListener("click",
            () => this.onClick());
    }

    onClick() {
        alert("Marker at " + this.element.style.left);
    }

    addTo(map) {
        map.appendChild(this.element);
    }
}
```

Using the Class

```
let m1 = new Marker(100, 150);
let m2 = new Marker(200, 250);

m1.addTo(map);
m2.addTo(map);
```

Now we think architecturally.

7 Reflection & Next Steps

Integrating Everything Together

Reflection

🤔 Why is WebGIS event-driven?
🤔 Why is everything modeled as objects?
🤔 What problem does OOP solve?
🤔 What do GIS libraries abstract away?

Key Takeaways

1. WebGIS ≠ just HTML
2. JavaScript controls **all interaction**
3. DOM enables **dynamic rendering**
4. Events **drive behavior**
5. OOP **structures complexity**

Today → Understanding the engine

Next → Using the engine

📋 Mini Assignment

Extend your project:

1. Add **3 different marker colors** (use a marker type parameter)
2. Add a **"Clear All Markers"** button
3. **Refactor** marker logic fully into a class
4. Add a **simple layer visibility toggle**
5. Or do other **creative things**
6. **Deploy** it (e.g. use GitHub Pages, netlify, ...)

Final Thought

JavaScript in WebGIS is **not about syntax**.

It is about:

Interactive Spatial System Architecture

📚 Further Reading & References

- MDN: DOM Introduction
- MDN: OOP in JavaScript
- This presentation and example

> Feel free to explore and bring questions to the next session!