Application Infrastructure Recap

Project Name: Gestionnaire Inspections Municipales (Urbops)
Architecture Style: Monolithic Frontend + Node.js Backend + Federated PostgreSQL Database System.

1. High-Level Architecture

The application is designed as a specialized tool for urban planning inspectors. It separates concerns into a **User-Centric Frontend**, a **Routing Backend**, and a **Federated Database Layer** that isolates different domains (Industry, Habitation, etc.) while maintaining a central "Brain" (Hub).

```
graph TD
    User((User/Inspector)) -->|HTTPS| Frontend[React Frontend]
    Frontend -->|REST API| Backend[Node.js Server]

    subgraph "Federated Database System"
        Backend -->|Writes Core Data| Hub[(City Hub)]
        Backend -->|Writes Specific Data| Spokes

        Hub -->|FDW Link| Geo[(City Geo Ref)]
        Hub -->|FDW Link| Codes[(City Codes)]

        subgraph Spokes [Domain Databases]
            Ind[(City Industrie)]
            Hab[(City Habitation)]
            Com[(City Commerce)]
            Perm[(City Permis)]
        end

        Hub -.->|Foreign Data Wrapper| Spokes
    end
```

2. Frontend (The Client)

Built with **React** and **Vite** for performance.

- **Tech Stack:** React 19, TailwindCSS, Lucide Icons, Leaflet (Maps).
- **State Management:** Context API (`AuthContext`, `InspectionContext`).
- **Key Components:**
    - **`InspectionMap.jsx`**: Visual interface using React-Leaflet to display lots and inspection statuses.
    - **`InspectionGrid.jsx`**: Dynamic form generation based on inspection type.
    - **`AdminPanel.jsx`**: User management (Create/Delete inspectors).
- **Authentication:** JWT-based. Tokens stored in `localStorage`. Session validated via `/api/auth/verify`.

3. Backend (The Router)

A **Node.js/Express** server that acts as a traffic controller.

- **Entry Point:** `server/index.js`.
- **Authentication:** `server/auth_controller.js` handles Login/Register using `bcryptjs` and `jsonwebtoken`.
- **Smart Routing (`db_router.js`):**
    - The server determines *where* to save data based on the `inspection_type`.
    - Example: An "Industrie" inspection saves core metadata to `city_hub` but detailed form data to `city_industrie`.
- **Diagnostics:** Includes self-healing and check scripts (`diagnostic_scan.js`, `verify_federation.js`).

4. The Federation (The Database)

The system uses **PostgreSQL Foreign Data Wrappers (FDW)** to treat 9 separate databases as one logical unit.

Database Tier	Database Name	Purpose
Tier 1: The Brain	`city_hub`	Authentication, Audit Logs, Central Inspection Registry (`inspections_hub`).
Tier 2: The Truth	`city_geo_ref`	Official Land Registry (Lots, Owners, Addresses).

	`city_codes`	Zoning laws, Usage codes, Regulatory rules.
Tier 3: The Spokes	`city_industrie`	Stores specific inspection details for Industrial zones.
	`city_habitation`	Residential inspection data.
	`city_permis`	Permit issuance and tracking.
	(+4 others)	Commerce, Recreation, Public, etc.

Key Mechanism: The "Virtual Codebase"

- **Schema Reusability:** 7 SQL files create 9 Databases. For example, `01_industrie.sql` is reused for `city_commerce_service` and `city_public_institutionnel` because the data structure is identical, preventing schema drift.
- **Cross-Querying:** The Hub can query `SELECT * FROM foreign_city_industrie.inspection_details` as if it were a local table, enabling unified reporting without monolithic complexity.

5. Security & Maintenance

- **Auth:** Role-Based Access Control (RBAC) with `Admin` and `Inspector` roles.
- **Linting:** Standardized via `eslint.config.js` (Server & Build scripts ignored).
- **Diagnostics:** Automated Health Checks verify API uptime and DB connectivity (< 450ms latency is normal for spoke connection).