Logistics simulation graph specification

1. Purpose

This document defines the structure, rules, and processing logic for generating and using a graph-based logistics simulation in a post-apocalyptic logistics system. The graph represents a hybrid network of **fulfillment centers** and **markets**, connected by various types of paths (road, rail, etc.), where each node and connection carries semantic weight impacting routing algorithms.

2. Node Types

Each node in the graph is either a **fulfillment center** or a **market**, and connects to the same type:

2.1 Fulfillment Center

- Holds inventory.
- Participates in routing only for **Ford-Fulkerson** and **Kaufmann-Malgrange** algorithms.

2.2 Market

- Doesn't hold inventory.
- Participates in routing using **Bellman-Ford** algorithm.

3. Node Structure

Each node is represented as a structured object with the following properties:

```javascript
{
  "node_id": "N001",
  "node_name": "East Fulfillment Center",
  "node_type": "fulfillment_center", // or "market"
  "node_description": "Short narrative for human identification",
  "node_location": {
```

```
    "latitude": 40.7128,
    "longitude": -74.0060
  },
  "node_tags": ["food", "ammo"],
  "is_secure": true,
  "is_active": true,
  "connections": [
    {
      "to": "N002",
      "connection_type": "road",
      "connection_conditions": ["infected_activity"]
    }
  ],
  "schema": {
    "version": 1.0
  }
}
```

3.1 Node Restrictions

- If `is_secure == false` **or** `is_active == false`, the node is treated as an **obstacle** and must not be included in any valid route or flow.
- Algorithms must **exclude non-secure or inactive nodes** from traversal or flow calculations.

4. Connection Properties

Each connection (edge) has the following properties:

- `to`: Target node ID.
- `connection_type`: See below.
- `connection_conditions`: List of environmental conditions affecting edge cost.

4.1 Connection Types and Base Modifiers

Type	Description	Base Modifier
`road`	Standard land routes, partially preserved.	`1.0`
`rail`	Train lines; faster, more reliable.	`0.7`
`trail`	Foot or bike paths; higher danger.	`1.3`
`tunnel`	Underground routes; secure, limited capacity.	`1.1`
`bridge`	May collapse or be unsafe.	`1.4`
`waterway`	Boat or raft access; fast but weather-dependent.	`0.9`
`drone`	Airborne supply drop; fast but limited payload.	`1.2`
`blocked`	Exists in metadata but not usable in pathfinding.	∞
`manual`	Human courier; very slow, low-tech fallback.	`1.6`

4.2 Environmental Condition Modifiers

Condition	Modifier	Notes
`infected_activity`	`+0.3`	Dangerous zone, risk of ambush or patrols
`weather_rain`	`+0.2`	Slippery, especially for trail and bridge
`foggy_visibility`	`+0.1`	Affects visibility, slows human movement
`cleared`	`-0.2`	Cleared by patrols or scouts
`reinforced`	`-0.3`	Infrastructure reinforced or maintained

4.3 Final Edge Cost Formula

```
C/C++

edge_cost = base_weight * connection_type_modifier *
node_B_security_modifier * Σ(condition_modifiers)
```

- `base_weight`: Predefined edge weight [1, 100].
- If either node is inactive or insecure, the edge must be excluded.

5. Routing Logic by Algorithm

5.1 Bellman-Ford (For Markets Only)

- Input: Weighted directed graph built from market nodes and connections.
- Output: Minimum-cost paths from a market to all others.
- Exclude: Fulfillment centers, inactive/insecure nodes, blocked or none connections.

5.2 Ford-Fulkerson (For Fulfillment Centers Only)

- Input: Flow graph of fulfillment center nodes.
- Output: Max resource flow.
- Capacity affected by path type and status.

5.3 Kaufmann-Malgrange (For Fulfillment Centers Only)

- Input: Secure, active fulfillment centers.
- Output: Hamiltonian path feasibility.
- Only usable on small graphs.

6. Validation Rules

- Unique `node_id` per node.
- Connections must target valid, unique nodes and must be treated as direct connections.
- Blocked or invalid nodes/connections excluded from algorithms.

7. Integration Targets

- Output as `.json` for consumption.
- **Algorithm output** is obtained by custom input as a **request**, based on the uploaded map.
- **REST endpoints** must support:
    - **POST** `/map` – Upload map in JSON schema format
    - **POST** `/request` – Submit algorithm request
        - **POST** `/markets-path` – **Bellman-Ford**
        - **POST** `/fulfillment-circuit` – **Ford-Fulkerson**
        - **POST** `/fulfillment-flow` – **Kaufmann-Malgrange**
    - **GET** `/results/` – Retrieve stored computation results
        - *Body*

```javascript
{
  "nodes": ["N001", "N002"]
}
```

- All algorithm results must be persisted in **MongoDB**, including metadata about the request and execution.