Pokemon Type Network Graph
System Design Document

1. Project Overview
This project models Pokemon Generation 1 type interactions as a directed, weighted network graph. The system is designed to support graph analysis and later 3D visualization, with a strong focus on learning and applying network theory concepts in a real domain.

2. Project Goals
- Learn graph theory through a real world domain
- Build a backend graph analysis service
- Explicitly store all interaction data
- Support filtered graph views
- Compute meaningful graph metrics
- Support future interactive 3D visualization

3. Non Goals
- No full UI implementation in MVP
- No Pokemon level modeling in MVP
- No persistent database
- No authentication or user accounts

4. Dataset Scope
The project starts with Pokemon Generation 1 only.

Types included:
Normal, Fire, Water, Electric, Grass, Ice, Fighting, Poison, Ground, Flying, Psychic, Bug, Rock, Ghost, Dragon

For every attacker to defender pair, the following is stored:
- Effectiveness multiplier (0, 0.5, 1, 2)
- Immunity flag for multiplier 0

5. Graph Model
Nodes: Each node represents a Pokemon type.
Edges: Directed from attacker to defender.
Weight: Effectiveness multiplier.
Immunity: Stored as a special edge attribute.

There is one node per type. Offense and defense are derived from edge direction.

6. Canonical Graph
A single canonical graph is built once at backend startup.
- Fully connected directed graph
- Includes neutral interactions
- Immutable and cached in memory
This graph acts as the single source of truth.

7. Graph Views and Filtering
Derived graph views are generated per request using filters:

- Offense only (outgoing edges)
- Defense only (incoming edges)
- Include neutral interactions
- Immunity only interactions

Immunity only overrides other filters.

8. Metrics

Metrics are computed on derived graph views.

Degree:
- Out degree measures offensive reach
- In degree measures defensive vulnerability

Weighted Centrality:
- PageRank style algorithm
- Directed and weighted by multipliers
- Outgoing weights normalized per node

9. Backend Architecture

- FastAPI backend
- In memory canonical graph
- Stateless derived views per request

Backend Layers:
1. Data module (static Gen 1 data)
2. Graph service (build and cache graph)
3. View service (apply filters)
4. Metrics service (degree and centrality)
5. API layer

10. API Design

GET /graph
Returns nodes and edges for a filtered graph view.

GET /metrics
Returns per node degree and centrality metrics.

Optionally combined as GET /analysis.

11. Frontend Architecture (Planned)

- Next.js application
- Client side rendering for 3D visualization
- Uses Three.js based force graph libraries

Frontend responsibilities:
- Request graph data and metrics
- Render interactive 3D network
- Map metrics to visual properties (size, color, thickness)
- Provide filter toggles

12. 3D Visualization Approach

- Uses existing Three.js graph libraries

- Layout is physics based and visual only
- Graph meaning is conveyed through color, size, direction, and interaction
- No semantic meaning encoded in node distance

13. Data Flow

Startup:

- Load Gen 1 data
- Build canonical graph

Request handling:

- Parse filters
- Generate derived view
- Compute metrics
- Return JSON response

14. Testing Strategy

- Verify known type matchups
- Validate filter behavior
- Degree and centrality sanity checks
- API response consistency

15. Extension Roadmap

- Add later generations
- Introduce Pokemon nodes
- Add competitive usage statistics
- Add additional graph metrics