System Requirements Document (SRD) for Decision Node Graphing Tool

Version: 1.1
Date: [Insert Date]
Author: Timothy S-W

Change Log:

2/9/2025	1.1	**Key Updates Based on Decisions Made** ✔ **Edge weighting & confidence scores** – Helps quantify relationship strength. ✔ **Historical comparisons & version tracking** – Enables better decision analysis over time. ✔ **Collapsible subgraphs** – Improves navigation for large graphs. ✔ **Undo/redo functionality for edits** – Helps users manage changes. ✔ **Role-based access control (RBAC)** – Prepares for multi-user access. This updated **SRD v1.1** reflects all key design decisions discussed. 🚀 Would you like any additional refinements?
2/19/25	**1.2**	**Key Additions & Changes:** ✔ **New Use Cases:** Expanding/Collapsing Subgraphs, Graph Editing (Add/Remove Nodes & Edges), Exporting Graph Data, Multi-User Collaboration. ✔ **Extended Use Cases:** Confidence Scores now include Historical Tracking & Automated Adjustments. ✔ **New Functional Requirements:** Added subgraph control, graph editing, real-time collaboration, and export features. ✔ **New Non-Functional Requirements:** Included system logs, WebSockets for real-time updates, and consistency enforcement in multi-user environments. ✔ **Refined Requirement Mapping:** All use cases are now directly linked to specific system requirements.

---

1. Introduction

1.1 Purpose

The **Decision Node Graphing Tool** is designed to **visualize relationships** between decisions, impacts, topics, and sources. The system enables users to explore, analyze, and interact with structured decision-making data through an intuitive **graph-based interface**.

This tool will support policy analysts, researchers, and data scientists in **understanding decision impacts, historical trends, and relationships** between various entities.

1.2 Scope

The tool will:

- Store and validate node/edge data directly in Neo4j.
- Render a dynamic graph of decision-related entities.
- Provide search, filtering, and interactive exploration of nodes.
- Support real-time data updates via a refresh mechanism.
- Enable future expansion for API-based access, analytics, and role-based access control (RBAC).
- Support real-time multi-user collaboration and historical tracking of decisions.
- Allow users to expand/collapse subgraphs for better navigation.
- Enable users to edit graphs by adding/removing nodes and edges.
- Provide export capabilities in multiple formats (PNG, SVG, JSON, CSV).

1.3 Intended Audience

- **Policy Analysts** – Evaluating decision impacts.
- **Researchers** – Investigating historical decision patterns.
- **Data Scientists** – Analyzing trends in decision-making.
- **General Users** – Exploring interconnected decisions.

1.4 System Overview

This system will provide a **visual knowledge graph** with interactive capabilities to explore decisions and their relationships. Users will interact with a UI to **inspect node details, filter data, and update content dynamically**. The system will also allow for **historical comparisons** of decisions and their impacts.

---

2. Use Cases

Use Case 1: Searching for a Specific Node

Actors: Policy Analyst, Researcher, Data Scientist, General User
Goal: Locate and highlight a specific node based on a search query.
Preconditions:

- The system is running and displaying a graph.
- The database contains nodes with searchable attributes (e.g., title, tags, category).

Postconditions:

- The user sees the search result highlighted.
- The graph may be adjusted (zoomed in) for better visibility.

Basic Flow:

1. User enters a search term in the search bar.
2. System validates the input and searches for matching nodes.
3. If found, the matching nodes are highlighted.
4. If multiple matches exist, the system presents a selection list.
5. User selects a node from the results (if needed).
6. The graph view zooms in on the selected node.

Use Case 2: Filtering Nodes by Type

Actors: Policy Analyst, Researcher, Data Scientist, General User
Goal: Apply filters to display only relevant node types.
Preconditions:

- The system is running with a rendered graph.
- The user is familiar with the filtering options.

Postconditions:

- Only selected node types are displayed.
- Graph layout adjusts dynamically.

Basic Flow:

1. User selects a filtering option (e.g., show only "Decisions" and "Impacts").

2. System applies filters and hides non-relevant nodes.
3. User interacts with the filtered graph.
4. User can reset filters to restore the full graph.

Use Case 3: Viewing Node Details

Actors: Policy Analyst, Researcher, Data Scientist, General User
Goal: Retrieve metadata and properties of a specific node.
Preconditions:

- The system is running and displaying a graph.
- The user can interact with nodes.

Postconditions:

- The selected node's details are displayed in the UI.

Basic Flow:

1. User clicks or hovers over a node.
2. System fetches metadata (e.g., category, status, tags, confidence score).
3. System displays node details in a sidebar or tooltip.

Use Case 4: Updating Data and Refreshing the Graph

Actors: System Admin, Editor
Goal: Load updated decision data and apply changes to the graph.
Preconditions:

- The system is connected to the Neo4j database.
- The user has permission to update data.

Postconditions:

- The updated data is reflected in the graph.
- Validation ensures no invalid relationships or duplicates are introduced.

Basic Flow:

1. User uploads a new dataset or requests a refresh.
2. System validates data for missing IDs, duplicates, and invalid relationships.

3. If valid, system updates the graph.
4. If errors exist, system alerts the user.

Use Case 5: Assigning Confidence Scores to Relationships

Actors: Data Scientist, Policy Analyst
Goal: Define and apply confidence scores to edges representing relationships.
Preconditions:

- The system supports **edge weighting** and **timestamps**.
- Confidence scoring logic is defined.

Postconditions:

- Relationships are visually represented with confidence scores.
- The graph reflects these weightings.
- Users can track how relationships change over time.
- The system provides a **historical comparison** view.
-

Basic Flow:

1. User selects an edge to inspect.
2. System displays confidence_score and timestamp metadata.
3. User updates or verifies the confidence score.
4. System stores the update with a new timestamp.
5. User can toggle between historical versions of a relationship.

Use Case 6: Expanding/Collapsing Subgraphs

Actors: Policy Analyst, Researcher, Data Scientist, General User
Goal: Reduce visual clutter by expanding or collapsing specific graph sections.
Preconditions:

- The graph contains subgraph groupings.
- Users can toggle subgraph visibility.

Postconditions:

- The graph updates dynamically based on user interactions.

Basic Flow:

1. User selects a collapsible subgraph.
2. System collapses or expands related nodes.
3. Graph layout adjusts dynamically.

Use Case 7: Role-Based Access Control (RBAC)

Actors: System Admin, Editor, Viewer
Goal: Restrict user access based on roles.
Preconditions:

- The system enforces role-based access.
- Users have assigned roles.

Postconditions:

- Users can only access permitted functionalities.

Basic Flow:

1. User logs in with credentials.
2. System verifies role permissions.
3. User can perform actions based on their assigned role.

Use Case 7.1: Real-Time Multi-User Collaboration

Actors: Multiple Users (Editors, Viewers), System Admin
Goal: Support **simultaneous modifications** by multiple users in real-time.
Preconditions:

- Users have **proper access roles** (RBAC).
- System supports **real-time synchronization**.

Postconditions:

- Changes are reflected across all active user sessions.

Basic Flow:

1. **Multiple users** access the graph.

2. A user **makes an edit** (e.g., adds a node, updates an edge).
3. System **broadcasts the change** to all active users.
4. Other users see the update **in real-time**.
5. System handles **conflict resolution** if simultaneous edits occur.

Use Case 8: Expanding/Collapsing Subgraphs

Actors: Policy Analyst, Researcher, Data Scientist, General User
Goal: Allow users to expand or collapse related nodes to improve navigation and readability in large graphs.
Preconditions:

- The graph contains predefined subgraph structures.
- The UI has an interaction mechanism (e.g., toggle, right-click menu) to expand/collapse nodes.

Postconditions:

- Users can reduce visual clutter by collapsing unnecessary sections.
- Expanded nodes reveal deeper relationships without affecting overall layout.

Basic Flow:

1. User selects a node or a set of related nodes.
2. System displays an option to expand or collapse the subgraph.
3. If **expanded**, additional related nodes and edges appear.
4. If **collapsed**, the grouped nodes are hidden, and a placeholder represents them.
5. The system preserves expanded/collapsed states during navigation.

Use Case 9: Graph Editing (Add/Remove Nodes & Edges)

Actors: Policy Analyst, Researcher, Editor
Goal: Allow users to **manually** modify graph structure by adding or removing nodes and edges.
Preconditions:

- The user must have **editor privileges**.
- The system provides an interface for **graph modification**.

Postconditions:

- The graph updates dynamically with new/deleted nodes and edges.
- Edits are **validated** before being committed.

Basic Flow:

1. User selects **“Edit Graph”** mode.
2. User clicks to **add a new node**.
3. System prompts for metadata (e.g., node type, title).
4. User selects an existing node and **creates a new edge**.
5. System updates the graph and **validates the change**.
6. User can **remove** a node or edge.
7. System ensures dependencies are resolved before deletion.
8. Changes are **saved** with timestamps.

Use Case 10: Exporting Graph Data

Actors: Policy Analyst, Researcher, Data Scientist, General User
Goal: Export graph data in various formats (PNG, SVG, JSON, CSV) for external use.
Preconditions:

- The graph must be rendered in a **stable state**.
- The system provides export options.

Postconditions:

- Users obtain **downloadable files** in the selected format.

Basic Flow:

1. User selects **“Export”** from the menu.
2. System provides export format options.
3. User selects a format (PNG, SVG, JSON, CSV).
4. System generates and downloads the file.

Functional Requirements

Requirement ID	Description	Use Case Steps Satisfied
FR-1	The system must store and manage nodes and edges within Neo4j.	Use Case 4: Steps 1, 2, 3

FR-2	The system must validate data consistency (e.g., missing IDs, invalid relationships).	Use Case 4: Steps 2, 3, 4
FR-3	The system must support indexing for frequently searched properties (tags, status, title).	Use Case 1: Step 2, 3
FR-4	The system must render a dynamic, interactive graph displaying nodes and edges.	Use Case 1: Step 5, Use Case 2: Step 3, Use Case 6: Step 3
FR-5	Nodes must be color-coded based on type (Decision, Impact, Topic, Source).	Use Case 2: Step 1, 2
FR-6	Users must be able to zoom, pan, and adjust layout dynamically.	Use Case 1: Step 6
FR-7	The graph must update when new data is imported or refreshed.	Use Case 4: Steps 3, 4
FR-8	Edges must be labeled with their relationship type (e.g., causes, references).	Use Case 5: Steps 1, 2
FR-9	Users must be able to click/hover over a node to view detailed metadata (e.g., status, category, tags).	Use Case 3: Steps 1, 2, 3
FR-10	The system must support searching for specific nodes using a search bar.	Use Case 1: Steps 1, 2, 3
FR-11	Users must be able to filter nodes by type, category, or other attributes.	Use Case 2: Steps 1, 2, 3
FR-12	The system must display edges with labeled relationships (e.g., causes, references).	Use Case 5: Steps 1, 2, 3
FR-13	Edge styles must distinguish different relationship types (e.g., different colors or thickness).	Use Case 5: Step 4

FR-14	The system must support edge weighting to indicate the strength of a relationship.	Use Case 5: Steps 3, 4
FR-15	The system must support confidence_score for relationships.	Use Case 5: Steps 2, 3, 4
FR-16	Users must be able to refresh the dataset and see real-time updates.	Use Case 4: Step 1, 2
FR-17	The system must validate updates before applying them to the graph.	Use Case 4: Step 2, 3
FR-18	Users must be alerted if data conflicts occur (e.g., duplicate IDs, conflicting relationships).	Use Case 4: Step 4
FR-19	The system must support timestamps (timestamp_created, timestamp_modified) for all relationships to track historical changes.	Use Case 5: Steps 3, 4
FR-20	Users must be able to expand and collapse subgraphs to reduce visual clutter.	Use Case 8
FR-21	Users must be able to manually add or remove nodes and edges.	Use Case 9
FR-22	The system must validate user modifications (e.g., ensure no orphaned nodes after deletion).	Use Case 9
FR-23	Users must be able to export graph data in PNG, SVG, JSON, or CSV format.	Use Case 10
FR-24	Users must be able to view historical versions of relationships for comparison.	Use Case 5
FR-25	The system must allow for **automated confidence score adjustments** based on supporting sources.	Use Case 5

FR-26	Users must be able to collaborate in real-time, seeing each other's updates.	Use Case 7
FR-27	The system must resolve conflicts when multiple users edit the same node or edge.	Use Case 7

Non-Functional Requirements (NFRs)

Requirement ID	Description	Satisfies Use Cases
NFR-1	The graph must render within 5 seconds for datasets up to 5,000 nodes.	Use Case 1, 2
NFR-2	Searching for a node should take no longer than 1 second.	Use Case 1
NFR-3	The system should support transitioning to an API backend in future versions.	Use Case 4
NFR-4	The system must handle growing datasets efficiently with optimizations (e.g., lazy loading, pagination).	Use Case 1, 2, 4
NFR-5	The UI must be intuitive and accessible to non-technical users.	Use Case 3, 6
NFR-6	Tooltips or help documentation must be available for guidance.	Use Case 3
NFR-7	If user authentication is added, the system must enforce **Role-Based Access Control (RBAC)**.	Use Case 7
NFR-8	Sensitive data (if any) should be encrypted.	Use Case 7

NFR-9	Users with different roles (e.g., viewer vs. editor) should have distinct permissions.	Use Case 7
NFR-10	**RBAC must restrict user permissions** based on role levels.	Use Case 7
NFR-11	**Bias Transparency**: Any bias rating property must be linked to a clear function for transparency.	Use Case 5
NFR-12	All imported data must be checked for duplicate entries, invalid relationships, and missing attributes.	Use Case 4
NFR-13	**Real-time multi-user collaboration must maintain consistency across user sessions.**	Use Case 7
NFR-14	The system must use **WebSockets or equivalent** for real-time updates.	Use Case 7
NFR-15	System logs must store **modification history** for debugging and tracking.	Use Case 5, 7

-

---

4. Future Enhancements

- **Collapsible Subgraphs** – Allow users to expand/collapse related nodes to reduce visual clutter.
- **Version Control** – Track changes in decisions and impacts over time for historical analysis.
- **Graph Editing** – Enable users to manually add/remove nodes and edges.
- **Export Options** – Allow users to download graphs as **PNG, SVG, JSON, or CSV**.
- **API Integration** – Enable external systems to query decision data dynamically.
- **Multi-User Collaboration** – Allow multiple users to interact with the same graph in real-time.
- Automated Confidence Score Adjustments – Algorithm to update confidence scores based on supporting sources.
- Advanced Indexing – Improve query execution speed through optimized database indexing.

- Bias Transparency in UI – Allow users to explore how bias_rating is determined through an interactive UI element.
- Collapsible Subgraphs – Allow users to expand/collapse related nodes to reduce visual clutter.
- Version Control – Track changes in decisions and impacts over time for historical analysis.
- Graph Editing – Enable users to manually add/remove nodes and edges.
-

---

5. Next Steps

1️⃣ **Implement Neo4j integration, replacing CSV storage.**
2️⃣ **Define confidence_score algorithm for evaluating relationships.**
3️⃣ **Ensure indexing is applied to improve search performance.**
4️⃣ **Update UI to incorporate new properties (summary, population_affected, bias_rating).**

6. Appendix

6.1 Data Dictionary Reference

- See the **Data Dictionary** for field definitions and relationships.