Here is a Product Requirements Document (PRD) for a comprehensive front-end for your **Infinite Concept Expansion Engine (Continuum)**.

This PRD is based on the analysis of your repository, which describes a powerful, autonomous, and self-learning system that expands concepts into a multimodal knowledge web. The goal of this front-end is to provide an intuitive, visual, and interactive layer to make this complex backend system accessible and easy to use.

Product Requirements Document: Continuum Visual Interface

1. Introduction

Continuum is a sophisticated autonomous AI system designed for infinite knowledge exploration. It takes seed concepts and dynamically expands them into a vast, interconnected knowledge graph, enriched with multimodal content (text, images, audio, video).

Currently, the system's power is primarily accessible via its REST API. This PRD outlines the requirements for a comprehensive front-end (web application) that will serve as the primary user interface for interacting with, visualizing, managing, and monitoring the Continuum engine. This "missing front-end" will unlock the system's value for a broader audience and make its complex operations intuitive and engaging.

2. Goals & Objectives

The primary goal of this front-end is to **make the Continuum engine's power simple to use and visually compelling.**

- **For Users:** Provide an intuitive way to initiate, explore, and interact with the knowledge graphs generated by the engine.
- **For Operators/Admins:** Offer a clear dashboard to monitor the system's health, learning progress, and resource consumption.

- **For the Product:** Transform an API-first backend into a complete, user-facing product that visually demonstrates its unique "self-improving" and "infinite expansion" capabilities.

3. Target Users

1. **The "Explorer" (Primary User):** Researchers, students, creatives, or strategists who want to input a starting concept (e.g., "Quantum Computing," "Renaissance Art," "Supply Chain Logistics") and explore the vast, interconnected web of information the system generates.
2. **The "Curator" (Admin/Power User):** The owner of the app (you) or a technical user who needs to configure the system, monitor its performance, manage API keys (for LLMs), and observe its learning patterns.

4. Key Features (Functional Requirements)

This front-end will be structured around several key modules:

Module 1: The Main Dashboard ("Control Center")

This is the user's home screen and central hub.

- **Concept Input:** A prominent, simple input bar: "Start exploring a new concept..."
- **My Expansions:** A gallery or list view of previously generated "concept webs." Each card should show:
  - The root concept (e.g., "Eiffel Tower").
  - A snapshot/thumbnail of the knowledge graph.
  - Key stats: # of nodes, # of images/videos, last updated.
- **System Status:** A simple widget showing the system's current status (e.g., "Idle," "Expanding Concept X," "Learning...").
- **Quick Access:** Buttons to navigate to System Monitoring and Settings.

Module 2: The "Continuum" Explorer (Visual Knowledge Graph)

This is the core feature and where users will spend most of their time. It's the visual representation of the "vast, interconnected web."

- **Interactive 3D/2D Graph:**
  - Use a graph visualization library (e.g., D3.js, three.js, react-flow) to render the knowledge graph.
  - The root concept should be at the center.
  - Related concepts (nodes) are connected by lines (edges).
  - The system should support **live updates**, showing new nodes and connections as the backend engine generates them.
- **Node Interaction:**
  - **On-Click:** Clicking a node opens a "Details" sidebar.
  - **On-Hover:** Shows a tooltip with a brief summary of the concept.
- **Details Sidebar:** When a node is selected, this panel appears, showing:
  - **Generated Text:** The detailed text content for that concept.
  - **Multimodal Gallery:** A carousel or grid displaying the generated images, videos, and audio files associated with that node.
  - **Metadata:** Source links, creation date, etc.
- **Exploration Controls:**
  - Zoom in/out, pan, and rotate (for 3D) the graph.
  - **"Expand this node":** A button on a selected node to manually trigger the backend to expand that specific concept further.
  - **Search:** A search bar to find specific nodes within the current graph.

Module 3: System Monitoring & Analytics (For the "Curator")

This is a read-only dashboard to visualize the "Real-time Monitoring" and "Persistent Learning" features of the backend.

- **Live Metrics:** A dashboard (using a library like Chart.js) that visualizes data from the /monitoring endpoints.
  - **Performance:** API response times, resource usage (CPU/memory), LLM token usage.
  - **Learning:** A chart showing the system's "improvement" or "confidence" score over time (from the feedback system).
  - **Content:** A donut chart showing the breakdown of generated content (e.g., % text, % image, % video).
- **Event Log:** A real-time, scrolling log of key system actions (e.g., "Started expansion for

'AI'," "Generated 5 new nodes," "Feedback loop initiated").

Module 4: Configuration & Settings

A simple "settings" page for the admin to manage the engine.

- **API Key Management:** A secure form to input and update API keys for the various LLM services (OpenAI, Anthropic, Google, etc.).
- **Engine Controls:**
  - Toggle switches to enable/disable specific content generators (e.g., "Disable Audio Generation").
  - Sliders or input fields to set parameters like "Max Expansion Depth" or "Expansion-per-minute."
- **Data Management:** A button to "Clear Cache" or "Prune old graphs."

---

5. Non-Functional Requirements

- **Usability:** The interface must be clean, modern, and intuitive. The complexity of the backend should be hidden behind a simple and delightful UI.
- **Performance:** The knowledge graph visualization must remain smooth and responsive, even with hundreds of nodes. Use lazy loading for multimodal content.
- **Responsiveness:** The application should be usable on desktop and tablet devices.
- **Tech Stack (Suggestion):**
  - **Front-End:** React, Vue, or Svelte.
  - **API Communication:** Use fetch or axios to communicate with your FastAPI backend.
  - **State Management:** Redux, Zustand, or Vuex to manage complex UI state.
  - **Visualization:** D3.js (for custom 2D) or Three.js (for 3D).

6. Success Metrics

We will know this front-end is successful when:

- **Engagement:** Users are initiating new concept expansions and spending time navigating the knowledge graphs.
- **Usability:** A new user can successfully start an expansion and find a piece of generated content with zero training.

- **Performance:** The graph explorer maintains a high frame rate (>30fps) even with a 500-node graph.
- **Adoption:** You (the developer) find yourself using the front-end for testing and demos instead of using API calls directly.