Devfolio Problem Statement 1: Vector + Graph Native Database for Efficient AI Retrieval

1. Problem Statement

Build a minimal but functional **Vector + Graph Native Database** that supports hybrid retrieval for AI applications. The system should ingest structured and unstructured data, store it as interconnected graph nodes enriched with vector embeddings, and expose a clean API for hybrid search, CRUD operations, and relationship traversal. The final project must run locally, be fast enough for real-time queries, and demonstrate how hybrid retrieval improves relevance over vector-only or graph-only search. The system should not use any such solutions that are specifically solving for the problem statement.

2. Background & Why It Is Needed

Modern AI systems depend heavily on retrieval for grounding, reasoning, and context stitching. Traditional vector databases struggle with deep relational queries. Graph databases excel at relationships but fail at semantic similarity. In real applications—knowledge assistants, personal knowledge graphs, enterprise RAG pipelines—you need both.

A Vector+Graph hybrid gives you:

- Semantic similarity through embeddings.
- Relationship reasoning through a graph.
- Better ranking, filtering, and multi-hop question answering.

Teams will design the foundation of such a system, proving that hybrid architectures can outperform single-mode retrieval.

3. Requirements

Your system must include:

- **Vector storage** with cosine similarity search.
- **Graph storage** with nodes, edges, and metadata.
- **Hybrid retrieval** that merges vector similarity + graph adjacency.
- **API endpoints** for CRUD, vector search, graph traversal, and combined search.
- **Simple scoring/ranking mechanism** for hybrid results.
- **Embeddings pipeline** (any open-source embedding model or a mocked vector generator).
- **Local persistence** (file-based, SQLite, or in-memory with snapshotting).

Stretch goals (optional):

- Multi-hop reasoning query.
- Relationship-weighted search.
- Basic schema enforcement.
- Pagination and filtering.

4. What You Have to Build

Participants must build a working prototype that:

- Ingests textual or structured input.
- Generates or accepts embeddings.
- Stores the input as graph nodes with properties.
- Links nodes using typed relationships.
- Allows querying: vector-only, graph-only, and hybrid.
- Provides ranking logic that combines semantic similarity and graph closeness.

- Demonstrates a real use-case dataset (notes, wiki snippets, research docs, scraped data , project data, etc.).

The deliverable should include:

- Backend service (any language).
- Minimal UI or CLI for showing queries.
- API documentation.
- A short demo explaining how they fulfill all the evaluation criteria.

5. Example Endpoints (CRUD + Search)

These are reference examples. Teams may modify structure but must preserve functionality.

Node CRUD

- `POST /nodes`
  - Create a node with text, metadata, and optional embedding.
- `GET /nodes/{id}`
  - Return the node with its properties and linked relationships.
- `PUT /nodes/{id}`
  - Update node metadata or regenerate embeddings.
- `DELETE /nodes/{id}`
  - Remove node and all associated edges.

Relationship CRUD

- `POST /edges`

    - Create a relationship: `{source, target, type, weight}`.
- `GET /edges/{id}`
    - Return edge details.

Vector Search

- `POST /search/vector`
    - Body: `{query_text, top_k}`.
    - Returns ranked matches by cosine similarity.

Graph Traversal

- `GET /search/graph?start_id=...&depth=...`
    - Returns reachable nodes up to specified depth.

Hybrid Search

- `POST /search/hybrid`
    - Body: `{query_text, vector_weight, graph_weight, top_k}`.
    - Returns merged scores + ranked output.

6. Evaluation Criteria

Evaluation happens in **two rounds**.

Round 1: Technical Qualifier

Scored on a 50-point scale.

- **Core functionality (20 pts):** Working CRUD, vector search, and graph traversal.

- **Hybrid retrieval logic (10 pts):** Scoring clarity, output relevance.
- **API quality (10 pts):** Clean structure, clear documentation.
- **Performance & stability (10 pts):** Fast enough for live demos.

Teams that score 35+ advance.

Round 2: Final Demo & Judging

Scored on a 100-point scale.

- **Real-world demo (30 pts):** Use-case clarity, working end-to-end flow.
- **Hybrid search effectiveness (25 pts):** Demonstrated improvement over vector-only/graph-only.
- **System design depth (20 pts):** How well they justify architecture, indexing, scoring.
- **Code quality & maintainability (15 pts):** Structure, readability, modularity.
- **Presentation & storytelling (10 pts):** Clarity, confidence, user understanding.

The highest-scoring team wins.

Note: The Core Members of OSC and AI/ML Club of Scaler School of Technology should have complete access(read and clone) to the codebase being made in and during the DevForge hackathon. This is needed so that the evaluators can go through your codebase in case of any discrepancy during evaluation. Failure to do so will result in the disqualification of the entire team.