Product Requirements Document (PRD): Echo

Project Name: Echo
Tagline: Connection without Performance
Hackathon Category: Mental Wellbeing & Community Connection (Problem Statement 2)
Timeline: 48 hours
Core Stack: Next.js, Tailwind CSS, PostgreSQL (pgvector), Google Gemini API

Architecture Note:
Echo intentionally uses a single PostgreSQL instance with **pgvector** to handle both relational data (users, entries) and semantic search (emotional similarity). Platforms like **Supabase** or **Neon** enable rapid setup, making this approach ideal for a 2-day hackathon.

---

1. Executive Summary

Echo is a *silent* social network designed to reduce isolation without the performative pressure or toxicity of traditional social media.

Instead of posting text, users vent privately. Their words are transformed by AI into abstract **Emotion Nodes**—floating visual blobs defined by color and shape. These nodes appear in a shared stream organized **by emotional similarity rather than chronology**, using vector embeddings.

Interaction is deliberately non-verbal. Users acknowledge one another through **Resonance**—a subtle light pulse—prioritizing **presence over performance**.

---

2. Problem Statement

The Problem

- Expressing pain online often invites judgment, toxic positivity, or silence.
- Many people want to be *seen*, not responded to or fixed.

The Solution

- A platform that enables:
    - **Externalization:** safely getting emotions out
    - **Internalization:** feeling understood without explanation

The Technical Insight

- Using **pgvector** to create *Semantic Gravity*: entries with similar emotional meaning naturally cluster together, allowing users to encounter others who feel the same *kind* of pain.

---

3. User Stories (The Happy Path)

Story A: The Release (Input)

- **As** a burnt-out student
- **I want** to vent about exam stress without fear of judgment
- **So that** I can get the feeling out of my head

Experience:
The user types *"School sucks"*

User enters and is shown their artwork of a small spiky red blob

---

Story B: The Connection (Stream)

- **As** a user viewing the stream
- **I want** to see others who feel what I feel
- **So that** I experience the universality of my struggle

Behind the Scenes:
The "School" blob is organised to spawn next to other school related blobs of similar sentiment (eg "School was tiring today" or "My teachers were mean today")

Story C: The Resonance (Interaction)

- **As** a user
- **I want** to acknowledge someone else's pain without words
- **So that** we both feel safely connected

Action:
The user double-taps a nearby blob.
Both blobs glow briefly.

---

4. Technical Architecture

4.1 Database Strategy: The One-DB Rule

Echo uses **PostgreSQL + pgvector** as the single source of truth for both structured data and embeddings.

Table: `entries`

Column	Type	Description
id	UUID	Primary key
user_id	UUID	Anonymous user identifier
message	VARCHAR(284)	Message
embedding	Vector(768)	Semantic embedding from Gemini
created_at	TIMESTAMP	Used for cleanup and trends

Table: `resonances`

Column	Type	Description
id	UUID	Primary key
target_entry_id	UUID	Blob being acknowledged
actor_entry_id	UUID	Blob doing the acknowledging

4.2 AI Pipeline (Gemini)

Each submission triggers:

1. **Analysis Call**
    - Submit message + standard template
    - Extracts emotion attributes:
        - Color
        - Shape
        - Intensity (size)

        - Category
2. **Embedding Call**
    - Generates a 768-dimension semantic vector for similarity search

(These can be combined if supported.)

4.3 Semantic Gravity (Core Algorithm)

Goal:
Reveal emotional similarity through interaction rather than layout, keeping the mural visually calm and static.

Static Mural Layout

- *The community mural is a **fixed, non-moving canvas**.*
- *When a new entry is created:*
    - *Its blob spawns at a **random, available position** on the mural.*
    - *Once placed, the blob **never moves**.*
- *The layout itself does **not** encode semantic meaning. There is no clustering, drifting, or rebalancing over time.*

This ensures the mural feels stable, quiet, and non-performative.

Hidden Semantic Structure

- *Each entry still has a vector embedding stored in Postgres.*
  *Semantic similarity exists entirely **under the surface** and is not visually expressed by default.*
- *Users are not overwhelmed by obvious groupings or labels.*

Interaction-Based Similarity Reveal

- *When a user clicks their own blob:*
    1. *We query pgvector for the k nearest neighbors using cosine distance (`<=>`).*
    2. *Entries with low cosine distance are identified as semantically similar.*
       *These blobs softly **glow** on the mural.*
- *All other blobs remain unchanged.*

*This interaction briefly reveals a **constellation of shared emotion** without permanently altering the space.*

Design Rationale

- *Random placement avoids premature categorization or comparison.*
- *Random placement avoids the situation where there is a predominant emotion causing only a portion of the mural to be filled.*
- *Similarity is revealed **only when the user chooses to look for it**.*

Result:
The mural remains quiet and neutral, while semantic connection is discovered through intentional interaction rather than constant visual signaling.

---

5. Functional Requirements (MVP)

P0 — Must Have (Demo-Critical)

1. **Text-to-blob**
2. **Vector Storage**
   Vector embeddings stored in Postgres.
3. **Mural View**
   Canvas rendering 20–50 blobs.
4. **Resonance Interaction**
   Clicking a blob triggers a glow animation.
5. **Mock Data Seeding**
   Script to insert ~50 fake entries so clustering is visible during the demo.

P1 — Nice to Have (Polish & Differentiation)

1. **Community Pulse Chart**
   Stream-graph showing emotional trends per person. So they are able to see their individual emotions over time and there will be a summary when requested (e.g. you are sad 6 times this month and 70% of when u are sad is on Saturday and you are sad because of …)
2. **Voice Input**
   Web Speech API → transcription → analysis.
3. **Haptic Feedback**
   Subtle vibration on Resonance (mobile).

---

6. API Endpoints (Next.js App Router)

`POST /api/entry`

Input

{ "text": "School is making me sad" }

Process

1. Gemini analyzes text → identify emotion, intensity
2. Gemini compares emotion and intensity to template → determine `{color, shape}`
3. Gemini generates embedding → `[0.12, -0.4, ...]`
4. Insert entry into Postgres

Output

{ "id": "...", "color": "#FF0000", "shape": "spiky", "x": 120, "y": 340 }

`GET /api/stream`

Input

- Optional: `?user_vector=[...]`

Process

- If vector provided → cosine similarity search
- Else → return latest 50 entries

Output

[{ "id": "...", "x": 120, "y": 340, "color": "#FF0000", "shape": "spiky" }]

`POST /api/resonate`

Input

{ "target_id": "UUID" }

Process

- Record resonance
- Trigger optional socket / animation event

---

7. Cursor / AI Coding Prompt

Paste into Cursor or your IDE AI:

> We are building “Echo”, a Next.js mental wellbeing app using PostgreSQL + pgvector.
>
> **Database:**
> Write a Supabase/Postgres SQL migration that enables the vector extension and creates an `entries` table with a `vector(768)` column.
>
> **Frontend:**
> Create a `Mural` component using `react-p5` that fetches blobs from the API.
>
> **Positioning Logic:**
>
> 1. Define three canvas centers:
>   - Work (top-left)
>   - Relationships (bottom-right)
>   - Self (center)
> 2. Place blobs near their category center with random drift
> 3. Use `framer-motion` for gentle floating animations

---

8. Hackathon Shortcuts

- **Trend Analysis:** Hardcode pulse data in JSON (no real aggregation).
- **Auth:** No login. Generate a random user ID and store it in `localStorage`.
- **Demo Setup:** Build mobile-first, but demo on desktop resized to mobile for projection clarity.

If you want, I can also:

- Cut this down to a **1-page judge version**

- Rewrite it as a **pitch narrative**
- Or turn the technical parts into a **build checklist for your team**