Collabryx Phase 1: Comprehensive Technical Specification & Implementation Guide

1. Executive Summary & Technology Stack

Collabryx is a full-stack AI networking platform designed to connect students, fresh graduates, and early-stage founders based on semantic compatibility. This document serves as the absolute source of truth for the engineering team.

1.1 Core Technology Stack Decisions

To ensure maximum compatibility and ease of development, the following technologies have been selected as the **mandatory** stack. No alternatives (e.g., "Redux vs Zustand") are presented; these are the locked choices.

- **Frontend Framework:** Next.js 14+ (App Router)
- **Language:** TypeScript (Strict Mode enabled)
- **UI Library:** shadcn/ui (Radix Primitives + Tailwind CSS)
- **Styling:** Tailwind CSS + clsx + tailwind-merge
- **Form Management:** React Hook Form + Zod (Schema Validation)
- **State Management:**
  - **Server State:** TanStack Query (React Query) v5 - *Mandatory for all data fetching.*
  - **Client State:** Zustand - *For UI state (sidebar open/close) and Auth Session.*
- **Backend / Database:** Supabase (PostgreSQL 15+)
- **Auth:** Supabase Auth (SSR w/ PKCE flow)
- **AI/LLM:** OpenAI API (gpt-4o-mini for Assistant, text-embedding-3-small for embeddings)
- **Icons:** Lucide React
- **Date Handling:** date-fns

2. System Architecture & Data Flow

2.1 High-Level Architecture

1. **Client (Next.js):** Handles UI, Input Validation, and optimistic updates.
2. **Middleware:** Intercepts requests to validate Supabase Auth Sessions before rendering protected routes.
3. **Supabase Client:** Direct connection to Postgres for standard CRUD (Read/Write own profile, Read messages) protected by Row Level Security (RLS).

4. **Supabase Edge Functions:** Serverless TypeScript functions running on Deno. These handle privileged logic:
    - Generating Embeddings (requires OpenAI Key).
    - Semantic Matching logic (Cosine Similarity).
    - AI Assistant processing.
5. **Database:** Postgres with pgvector extension enabled.

3. Database Schema (PostgreSQL)

All tables exist in the public schema. UUIDs are used for primary keys.

3.1 Extensions

create extension if not exists vector;

3.2 Tables Definition

profiles

The core user identity.

- **id**: uuid (Primary Key, Foreign Key to auth.users.id - ON DELETE CASCADE)
- **email**: text (Synced from Auth)
- **full_name**: text
- **role**: text (Enum: 'student', 'founder', 'professional')
- **headline**: text (Max 140 chars)
- **bio**: text (Max 1000 chars)
- **skills**: jsonb (Array of strings: ["React", "Marketing"]. JSONB allows for future expansion like [{name: "React", level: 5}])
- **interests**: text[]
- **experience_level**: text ('beginner', 'intermediate', 'advanced')
- **goals**: text
- **availability_hours**: int2
- **profile_completion_score**: int2 (0-100)
- **onboarding_completed**: bool (Default: false)
- **avatar_url**: text
- **created_at**: timestamptz
- **updated_at**: timestamptz

profile_embeddings

Stores the vector representation of the user for AI matching.

- **id**: uuid (PK)

- **user_id**: uuid (FK to profiles.id)
- **embedding**: vector(1536) (Matches OpenAI text-embedding-3-small)
- **last_updated**: timestamptz

connections

Tracks established relationships.

- **id**: uuid (PK)
- **user_a**: uuid (FK profiles.id)
- **user_b**: uuid (FK profiles.id)
- **status**: text ('active', 'blocked')
- **created_at**: timestamptz
- *Constraint:* Ensure user_a < user_b to prevent duplicate rows for the same pair.

connection_requests

- **id**: uuid (PK)
- **sender_id**: uuid (FK)
- **receiver_id**: uuid (FK)
- **status**: text ('pending', 'accepted', 'rejected', 'ignored')
- **created_at**: timestamptz

messages

- **id**: uuid (PK)
- **conversation_id**: uuid (FK to a conversations table or calculated via composite key)
- **sender_id**: uuid (FK)
- **content**: text
- **is_read**: bool (Default false)
- **created_at**: timestamptz

assistant_threads

- **id**: uuid (PK)
- **user_id**: uuid (FK)
- **title**: text
- **created_at**: timestamptz

assistant_messages

- **id**: uuid (PK)
- **thread_id**: uuid (FK assistant_threads.id)
- **role**: text ('user', 'assistant')
- **content**: text
- **created_at**: timestamptz

4. Security & Row Level Security (RLS)

RLS is **Mandatory**. No table shall be "Public" writable.

1. **profiles**:
   - **SELECT**: Authenticated users can read basic fields (id, full_name, headline, skills, avatar_url) of ALL users.
   - **UPDATE**: auth.uid() = id.
   - **INSERT**: Triggered by System (via Database Trigger on auth.users creation).
2. **profile_embeddings**:
   - **SELECT**: Service Role Only (Edge Functions). Users cannot read vectors directly.
   - **UPDATE/INSERT**: Service Role Only.
3. **messages**:
   - **SELECT**: auth.uid() IN (sender_id, receiver_id).
   - **INSERT**: auth.uid() = sender_id.

5. AI Implementation Details

5.1 Embedding Generation Strategy

Model: text-embedding-3-small (OpenAI).
Trigger: Invoked when the user completes Onboarding or updates their "Bio", "Skills", or "Goals".
Input Construction (The "Prompt" for Embedding):
Do not just embed the JSON. Create a semantic string:
"Role: Student. Headline: React Developer seeking Fintech startup. Bio: Computer Science junior passionate about AI and finance. Skills: React, Python, Finance. Goals: Build an MVP."

This string is sent to the OpenAI API. The resulting vector is stored in profile_embeddings.

5.2 Matching Logic (Edge Function: get-matches)

1. **Input:** Current User ID.
2. **Process:**
   - Fetch current user's embedding.
   - Perform a cosine similarity search against the profile_embeddings table.
   - Filter out users already in connections or connection_requests.
3. **SQL Query (inside Edge Function):**
   match_threshold := 0.5; -- Minimum similarity
   select profiles.id, profiles.full_name, 1 - (profile_embeddings.embedding <=> query_embedding) as similarity
   from profile_embeddings

```
join profiles on profiles.id = profile_embeddings.user_id
where 1 - (profile_embeddings.embedding <=> query_embedding) > match_threshold
order by similarity desc
limit 10;
```

5.3 AI Assistant (Edge Function: ai-assistant)

- **Model:** gpt-4o-mini (Fast, cheap, capable).
- **Streaming:** The Edge Function must return a ReadableStream to allow the text to type out on the frontend.
- **System Prompt:**"You are Collabryx, a startup mentor. You help students and founders refine ideas. Be concise, encouraging, and practical. Always suggest one concrete 'Next Step'."

6. Frontend Architecture & Directory Structure

We use the Next.js App Router with Route Grouping for organizational clarity.

```
/app
├── layout.tsx              # Global Layout (Providers: QueryClient, Theme, Auth)
├── globals.css             # Tailwind imports
├── (public)                # Route Group: Public access
│   ├── page.tsx            # Landing Page
│   ├── login/page.tsx      # Login Form
│   └── register/page.tsx   # Register Form
├── (auth)                  # Route Group: Protected (Middleware enforced)
│   ├── layout.tsx          # Authenticated Shell (Sidebar, Navbar, UserDropdown)
│   ├── dashboard/page.tsx  # Main feed
│   ├── onboarding/         # Multi-step form
│   │   ├── page.tsx
│   │   └── layout.tsx
│   ├── matches/page.tsx    # Swipe/List view of matches
│   ├── messages/           # Chat layout
│   │   ├── page.tsx        # "Select a conversation" placeholder
│   │   └── [id]/page.tsx   # Actual chat room
│   ├── assistant/page.tsx  # AI Chat interface
│   └── profile/
│       └── [id]/page.tsx   # View other profiles
├── api/                    # Next.js API Routes (if needed, prefer Server Actions)
│   └── auth/callback/route.ts # OAuth callback handler
```

6.1 State Management Rules

1. **Form Data (e.g., Onboarding):** Use react-hook-form with persist (localStorage) so users don't lose data on refresh, until submission.
2. **Remote Data (Profiles, Matches):** Use useQuery (TanStack Query).
   - Key: ['matches', userId]
   - Key: ['profile', profileId]
3. **Realtime Chat:** Use useEffect with supabase.channel to listen for INSERT on the messages table, then manually update the TanStack Query cache using queryClient.setQueryData.

7. Development Roadmap (Phase 1)

Sprint 1: Foundation & Auth

- **Task 1.1:** Setup Next.js + Supabase + Tailwind + shadcn.
- **Task 1.2:** Configure Database Tables (profiles, verification).
- **Task 1.3:** Implement (public)/login and (public)/register.
- **Task 1.4:** Setup Database Trigger: Create profiles row automatically when auth.users row is created.

Sprint 2: Profile & Onboarding

- **Task 2.1:** Build (auth)/onboarding wizard (3 steps: Basic Info, Skills, Goals).
- **Task 2.2:** Create Edge Function generate-embedding.
- **Task 2.3:** Wire onboarding completion to trigger the Edge Function.

Sprint 3: Matching Engine

- **Task 3.1:** Create Edge Function get-matches.
- **Task 3.2:** Build (auth)/matches UI.
- **Task 3.3:** Implement "Connect" button (Insert into connection_requests).

Sprint 4: Communication

- **Task 4.1:** Build Realtime Chat UI.
- **Task 4.2:** Implement (auth)/messages/[id].
- **Task 4.3:** Implement "Accept Request" logic.

Sprint 5: AI Assistant

- **Task 5.1:** Build Chat Interface for AI.
- **Task 5.2:** Create Edge Function ai-assistant with streaming response.
- **Task 5.3:** Inject User Context (Bio/Skills) into the System Prompt so the AI knows who it is talking to.

8. Specific Library Configurations

8.1 Supabase Client (lib/supabase.ts)

```
import { createBrowserClient } from '@supabase/ssr'

export function createClient() {
  return createBrowserClient(
    process.env.NEXT_PUBLIC_SUPABASE_URL!,
    process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!
  )
}
```

8.2 Zod Schema Example: Onboarding (lib/validations/onboarding.ts)

```
import * as z from "zod"

export const onboardingSchema = z.object({
  role: z.enum(["student", "founder", "professional"]),
  university: z.string().min(2, "University name is required"),
  skills: z.array(z.string()).min(3, "Select at least 3 skills"),
  bio: z.string().min(50, "Bio must be at least 50 characters").max(1000),
})
```

9. Deployment Strategy

9.1 Environment Variables

These must be set in Vercel (Production) and .env.local (Development).

```
# Supabase
NEXT_PUBLIC_SUPABASE_URL=[https://xyz.supabase.co](https://xyz.supabase.co)
NEXT_PUBLIC_SUPABASE_ANON_KEY=eyJ...
SUPABASE_SERVICE_ROLE_KEY=ey... # ONLY for Edge Functions

# OpenAI
OPENAI_API_KEY=sk-...

# App
NEXT_PUBLIC_APP_URL=http://localhost:3000
```

9.2 CI/CD

- **Platform:** Vercel.
- **Trigger:** Push to main.
- **Build Command:** next build.
- **Supabase:** Migrations must be applied manually or via GitHub Actions using supabase db push.