Collabryx Comprehensive Debugging & Observability Guide

1. Philosophy: "Log for Context, Not Just Errors"

In a complex stack involving AI (OpenAI), Database (Supabase), and Full-Stack Frameworks (Next.js), 90% of bugs are not "crashes" but "unexpected behaviors" (e.g., the AI gives a bad answer, or the match score is weirdly low).

The Golden Rule: Every major action must log its **Input** (what it received), its **Decision** (logic branches), and its **Output** (what it returned).

2. Global Debugging Standards

2.1 Structured Logging Format

Stop using console.log("here"). It is useless in production logs. Use a structured JSON-like format.

Bad:

```
console.log("User", user);
console.log("Error", err);
```

Good (The Standard):

```
console.log("[Auth-Middleware] Session Validated", { userId: user.id, role: user.role });
console.error("[Auth-Middleware] Validation Failed", { error: err.message, stack: err.stack });
```

2.2 Correlation IDs

When a user clicks "Find Matches", that request hits the Client -> API -> Edge Function -> Database -> OpenAI. You need to trace this path.

- **Action:** Generate a requestId at the start of the flow and pass it down.

3. Frontend Debugging (Next.js)

3.1 React Query (TanStack Query)

Since all data fetching is managed here, this is your first line of defense for "Missing Data" or

"Stale Data".

Tool: Install @tanstack/react-query-devtools.

npm i @tanstack/react-query-devtools

Integration:
Add this to your Providers component. It creates a floating icon in dev mode to inspect cache.

```
import { ReactQueryDevtools } from '@tanstack/react-query-devtools'

export function Providers({ children }) {
  return (
    <QueryClientProvider client={queryClient}>
      {children}
      <ReactQueryDevtools initialIsOpen={false} />
    </QueryClientProvider>
  )
}
```

3.2 Server Components vs. Client Components

- **Server Components (app/page.tsx)**: Logs appear in your **Terminal** where you ran npm run dev.
- **Client Components ('use client')**: Logs appear in the **Browser Console** (F12).

Strategic Log Placement:
In app/(auth)/dashboard/page.tsx (Server Component):

```
export default async function Dashboard() {
  const supabase = createClient();
  const { data: { user }, error } = await supabase.auth.getUser();

  // DEBUG: Check Auth state on server render
  if (error || !user) {
    console.error("[Dashboard] Server Auth Check Failed", error);
    redirect('/login');
  } else {
    console.log("[Dashboard] Rendering for User", user.id);
  }
  // ...
}
```

3.3 Middleware Debugging (middleware.ts)

Middleware runs before *every* request. If your app is stuck in a redirect loop, look here.

```
export async function middleware(request: NextRequest) {
  console.log(`[Middleware] ${request.method} ${request.nextUrl.pathname}`);

  const response = NextResponse.next({
    request: { headers: request.headers },
  });

  const supabase = createServerClient(..., { cookies: ... });
  const { data: { user } } = await supabase.auth.getUser();

  // DEBUG: verify session presence
  if (!user) {
    console.warn(`[Middleware] No user found for protected route:
${request.nextUrl.pathname}`);
  }

  return response;
}
```

4. Supabase & Database Debugging

4.1 Row Level Security (RLS) Inspection

If queries return empty arrays [] but no error, it is **always** RLS.

How to Debug:

1. Go to Supabase Dashboard > SQL Editor.
2. Run the query mimicking the user:

```
-- Impersonate the user to test RLS
set request.jwt.claim.sub = 'USER_UUID_HERE';
set role authenticated;

select * from profiles where id = 'TARGET_PROFILE_ID';
```

If this returns nothing, your RLS policy is too strict.

4.2 Database Triggers

Triggers (like auto-creating a profile) fail silently if not logged.

Action: Create a debug_logs table in Supabase.

```
create table debug_logs (
  id serial primary key,
  message text,
  payload jsonb,
  created_at timestamptz default now()
);
```

Inside your PL/PGSQL Function:

```
begin
  insert into public.profiles (id, email)
  values (new.id, new.email);
exception when others then
  -- DEBUG: Catch trigger failures
  insert into public.debug_logs (message, payload)
  values ('Profile Creation Failed', jsonb_build_object('error', SQLERRM, 'user_id', new.id));
  raise; -- Re-raise error so auth fails
end;
```

5. AI & Edge Function Debugging

Edge functions run in Deno. Logs here appear in the **Supabase Dashboard > Edge Functions > Logs**.

5.1 Function: generate-embedding

Critical Checkpoints:

1. Did we get the webhook payload?
2. Did the profile fetch succeed?
3. Did OpenAI accept the prompt?
4. Did the vector upsert work?

Code Instrumentation:

```
serve(async (req) => {
  const { record } = await req.json();

  // 1. Log Input
  console.log(`[Generate-Embedding] Triggered for User: ${record.id}`);

  // Check inputs
  if (!record.bio || record.bio.length < 10) {
    console.warn(`[Generate-Embedding] Skipped - Bio too short: "${record.bio}"`);
    return new Response(...);
  }

  // 2. Log Prompt Construction
  const prompt = `Role: ${record.role}...`; // truncated
  console.log(`[Generate-Embedding] Sending prompt to OpenAI (Length:
${prompt.length})`);

  try {
    const embeddingResponse = await openai.embeddings.create(...);

    // 3. Log Token Usage (Cost Management)
    console.log(`[Generate-Embedding] Success. Tokens used:
${embeddingResponse.usage.total_tokens}`);

    // 4. Log Vector Dimensions (Sanity Check)
    const vector = embeddingResponse.data[0].embedding;
    if (vector.length !== 1536) {
      throw new Error(`Vector dimension mismatch: Got ${vector.length}, expected 1536`);
    }

  } catch (err) {
    // 5. Catch OpenAI Errors
    console.error(`[Generate-Embedding] OpenAI/DB Error:`, err);
    return new Response(JSON.stringify({ error: err.message }), { status: 500 });
  }
});
```

5.2 Function: get-matches

Critical Checkpoints:

1. The embedding query itself.
2. The similarity score values (are they all 0.99? or all 0.10?).

Debugging SQL Logic:
In the SQL function match_profiles, add logging by raising notices (visible in Postgres logs) or returning extra debug columns during dev.
Frontend Hook Debugging (useMatches.ts):

```
const { data, error } = await supabase.rpc('match_profiles', { ... });

if (error) {
  console.error("[Get-Matches] RPC Failed", error);
} else {
  // DEBUG: Inspect Match Quality
  console.table(data.map(m => ({
    name: m.full_name,
    score: m.similarity.toFixed(4) // Check precision
  })));

  if (data.length === 0) {
    console.warn("[Get-Matches] No matches found. Threshold might be too high (0.5).");
  }
}
```

6. Local Development Debugging Flow

Do not push to Vercel to debug. Use the local environment.

1. **Start Supabase Locally:**
   npx supabase start

   This gives you a local Studio URL (usually http://127.0.0.1:54323). Go there to check database tables instantly.
2. **Serve Edge Functions Locally:**
   npx supabase functions serve --env-file .env.local --no-verify-jwt

   This allows you to hit http://localhost:54321/functions/v1/generate-embedding via Postman/Curl to test logic without touching the UI.
3. Mocking OpenAI (Optional but Recommended):
   If you are burning tokens debugging UI logic, temporarily mock the OpenAI response in your Edge Function:
   // DEBUG: Mock vector

```
// const embedding = new Array(1536).fill(0.1);
```

7. Troubleshooting Common "Collabryx" Errors

Scenario A: "I updated my bio, but matches didn't change."

Debug Steps:

1. Check profile_embeddings table in Supabase. Look at the last_updated column for your User ID.
   - *If old date:* The generate-embedding function didn't fire. Check Database Webhooks or Trigger logs.
   - *If new date:* The vector updated. The issue is likely the match_threshold in get-matches. Lower it from 0.5 to 0.1 to see if matches appear.

Scenario B: "The AI Assistant is hallucinating."

Debug Steps:

1. Go to app/api/chat/route.ts.
2. console.log("System Prompt:", systemMessage.content).
3. Verify the userContext injection. Is it passing empty skills?
   - *Fix:* If userContext is { skills: [] }, the AI invents skills. Ensure frontend fetches profile before opening chat.

Scenario C: "Supabase Auth session missing on refresh."

Debug Steps:

1. Check layout.tsx (Root). Are you wrapping the app in an Auth Listener?
2. Look at LocalStorage (Browser DevTools > Application > Local Storage). Do you see sb-<project-id>-auth-token?
   - *If no:* Login logic failed to persist.
   - *If yes:* Middleware might be stripping cookies. Check cookie settings in supabase.ts client creation.

8. Final Checklist Before "Done"

- [ ] **Error Boundaries:** Wrap your main components in React Error Boundaries to catch crashes and log them.
- [ ] **Network Tab:** Always keep the Network tab open. Look for 401 Unauthorized (Auth issues) or 500 Internal Server Error (Edge Function crashes).
- [ ] **Vector Health:** Periodically run select count(*) from profile_embeddings vs select count(*) from profiles. If they don't match, your sync is broken.