

OneMindAI: Vibe-Coding Control & Debug Guide

Document Version: 1.0

Created: December 12, 2025

Purpose: Solving the "Invisible Changes" Problem in AI-Assisted Development



TABLE OF CONTENTS

1. [THE PROBLEM: What's Going Wrong](#)
2. [THE SOLUTION: Admin-Controlled Configuration](#)
3. [WHY THIS IS FOOLPROOF \(And What Can Still Go Wrong\)](#)
4. [HARDCODED VS ADMIN: The Logic Explained](#)
5. [PROMPT 1: Controlled AI Changes \(Think Before Coding\)](#)
6. [PROMPT 2: Full System Story \(Code Flow Documentation\)](#)
7. [PROMPT 3: Live Debug Output \(Real-Time Execution Tracing\)](#)
8. [IMPLEMENTATION CHECKLIST](#)

● THE PROBLEM: What's Going Wrong {#the-problem}

Your Situation (In Plain English)

THE VIBE-CODING CHAOS

You prompt AI → AI changes code → Something breaks → You don't know what changed → You prompt again → More changes → More confusion

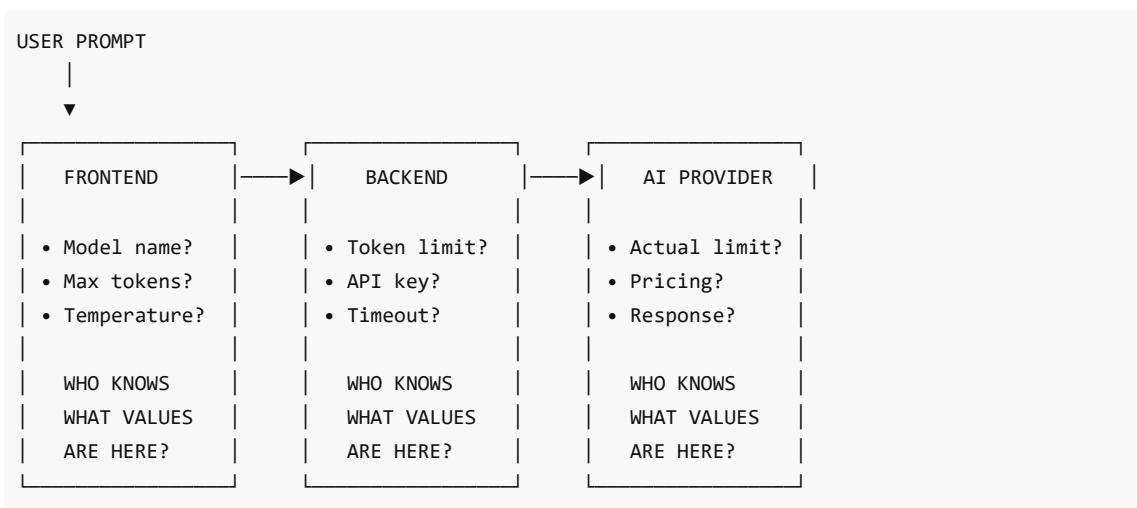
RESULT: A working system that nobody fully understands

The Specific Problems You Described

Problem	Example	Why It Happens
Invisible Changes	AI changes a token limit from 4096 to 8192 without telling you	AI focuses on "making it work" not "explaining what changed"
Frontend-Backend Mismatch	Frontend sends <code>max_tokens: 65536</code> , backend caps at 8192	Two different AI sessions wrote each part
Hardcoded Values Everywhere	Pricing in code, model names in code, URLs in code	AI takes the fastest path, not the maintainable path
No Data Visibility	You can't see what's being sent to APIs	No debug logging was requested

Config File Problem	To change a value, you edit code and redeploy	Values should be in database, not files
----------------------------	---	---

The Data Flow You Can't See



Your Question: "What is being passed from the first command to the API and back?"

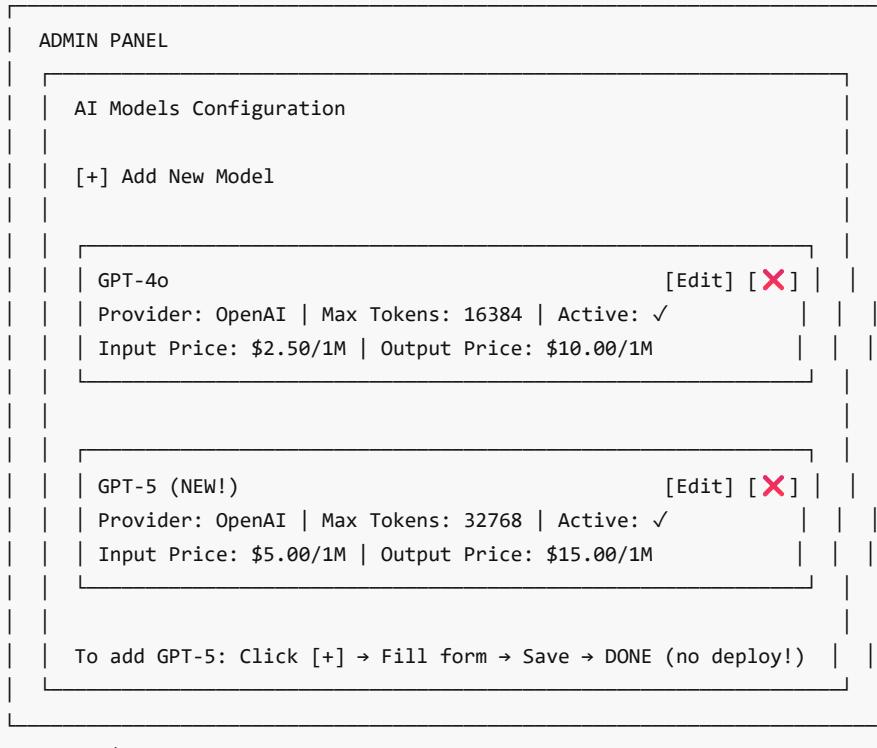
Current Answer: Nobody knows without reading every line of code.

THE SOLUTION: Admin-Controlled Configuration {#the-solution}

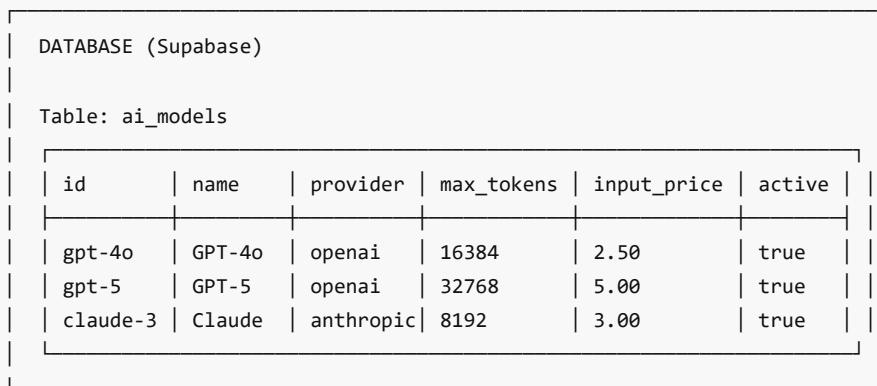
The Architecture Change

BEFORE (Hardcoded in Code)
<pre> OneMindAI.tsx (10,764 lines) const models = [{ id: 'gpt-4o', name: 'GPT-4o', maxTokens: 16384 }, ← HARDCODED { id: 'claude-3.5', name: 'Claude', maxTokens: 8192 }, ← HARDCODED]; const pricing = { 'gpt-4o': { input: 25, output: 100 }, ← HARDCODED }; To add GPT-5: Edit code → Commit → Deploy → Pray </pre>

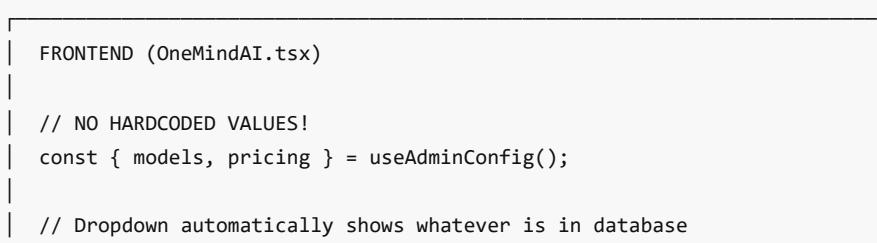
AFTER (Admin-Controlled Database)



Reads from database



Frontend fetches on load



```
| <Select options={models.filter(m => m.active)} />
```

Database Schema for Admin-Controlled Config

```
-- Table: ai_models (Admin controls this)
CREATE TABLE ai_models (
    id TEXT PRIMARY KEY,                      -- 'gpt-4o', 'claude-3.5-sonnet'
    display_name TEXT NOT NULL,                -- 'GPT-4o', 'Claude 3.5 Sonnet'
    provider TEXT NOT NULL,                   -- 'openai', 'anthropic', 'gemini'
    api_model_id TEXT NOT NULL,               -- Actual API model identifier
    max_tokens INTEGER DEFAULT 4096,
    default_temperature DECIMAL(3,2) DEFAULT 0.7,
    input_price_per_million DECIMAL(10,4),
    output_price_per_million DECIMAL(10,4),
    is_active BOOLEAN DEFAULT true,
    is_vision_capable BOOLEAN DEFAULT false,
    context_window INTEGER,
    description TEXT,
    display_order INTEGER DEFAULT 0,
    created_at TIMESTAMPTZ DEFAULT NOW(),
    updated_at TIMESTAMPTZ DEFAULT NOW()
);

-- Table: system_config (Admin controls this)
CREATE TABLE system_config (
    key TEXT PRIMARY KEY,
    value JSONB NOT NULL,
    description TEXT,
    updated_by UUID REFERENCES auth.users(id),
    updated_at TIMESTAMPTZ DEFAULT NOW()
);

-- Example system_config entries:
INSERT INTO system_config (key, value, description) VALUES
    ('rate_limits', '{"requests_per_minute": 60, "tokens_per_day": 1000000}', 'API rate limits'),
    ('default_max_tokens', '4096', 'Default max tokens if not specified'),
    ('markup_percentage', '30', 'Markup on provider costs'),
    ('signup_bonus_credits', '100', 'Credits given to new users');
```

What Changes in the Code

```
// BEFORE: Hardcoded in OneMindAI.tsx
const BASE_PRICING = {
    openai: {
        'gpt-4o': { input: 25, output: 100 },
        'gpt-4o-mini': { input: 1.5, output: 6 },
    },
}
```

```

// ... 200 more lines of hardcoded values
};

// AFTER: Fetched from database
const { models, pricing, config } = useAdminConfig();

// The hook fetches from Supabase on app load
function useAdminConfig() {
  const [models, setModels] = useState([]);
  const [config, setConfig] = useState({});

  useEffect(() => {
    const fetchConfig = async () => {
      const { data: modelsData } = await supabase
        .from('ai_models')
        .select('*')
        .eq('is_active', true)
        .order('display_order');

      const { data: configData } = await supabase
        .from('system_config')
        .select('*');

      setModels(modelsData);
      setConfig(Object.fromEntries(configData.map(c => [c.key, c.value])));
    };

    fetchConfig();
  });

  // Real-time updates when admin changes values
  const subscription = supabase
    .channel('config_changes')
    .on('postgres_changes', { event: '*', schema: 'public', table: 'ai_models' },
      fetchConfig)
    .on('postgres_changes', { event: '*', schema: 'public', table: 'system_config' },
      fetchConfig)
    .subscribe();

    return () => subscription.unsubscribe();
  }, []);

  return { models, config };
}

```



WHY THIS IS FOOLPROOF (And What Can Still Go Wrong) {#foolproof-analysis}

Why This Solution Works

Benefit	Explanation
No Code Changes for New Models	Admin adds GPT-5 in panel → Frontend automatically shows it
No Deployment Needed	Database change = instant update (real-time subscription)
AI Can't Accidentally Change Values	Values are in database, not code files AI edits
Single Source of Truth	One place for all configuration
Audit Trail	Database tracks who changed what and when
Rollback Possible	Database backups allow reverting changes

What Can Still Go Wrong

Risk	Likelihood	Mitigation
AI changes the hook code	Medium	Add comment: // DO NOT MODIFY - Admin controlled
AI adds hardcoded fallbacks	High	Use linting rules to detect hardcoded values
Database schema changes	Low	Lock schema, only DBA can modify
Admin enters wrong values	Medium	Add validation in admin panel
Real-time sync fails	Low	Add fallback to cached values
AI creates duplicate config system	Medium	Document architecture clearly

How to Prevent AI From Breaking Admin Values

```
// Add this comment block at the top of any file that uses admin config
/**
 * ⚠️ ADMIN-CONTROLLED CONFIGURATION ⚠️
 *
 * DO NOT hardcode any of the following values:
 * - Model names, IDs, or versions
 * - Pricing (input/output costs)
 * - Token limits
 * - Rate limits
 * - Feature flags
 *
 * All these values come from the database via useAdminConfig().
 * To change them, use the Admin Panel → Configuration.
 *
 * If you need a new configuration option:
 * 1. Add it to the system_config table
 * 2. Update the useAdminConfig hook to expose it
 * 3. Use it via config.yourNewOption
 */
```

```
* NEVER add hardcoded values to this file.  
*/
```

🎯 HARDCODED VS ADMIN: The Logic Explained {#hardcoded-vs-admin}

The Decision Framework

WHAT GOES WHERE?

Ask yourself: "Will a non-developer ever need to change this?"

YES → Put in ADMIN PANEL (database)

NO → Can stay in CODE (but consider config file)

Complete Classification

🔴 MUST BE IN ADMIN PANEL (Database)

Category	Examples	Why
AI Models	GPT-4o, Claude 3.5, Gemini 2.0	New models launch frequently
Pricing	\$2.50/1M input tokens	Providers change prices
Token Limits	Max 16384 tokens	May need adjustment per model
Feature Flags	Enable/disable providers	Business decisions
User Roles	CEO, CFO, Sales prompts	Marketing may want changes
Rate Limits	60 requests/minute	May need tuning
Signup Bonus	100 credits	Promotional changes
Markup Percentage	30% over cost	Business decision

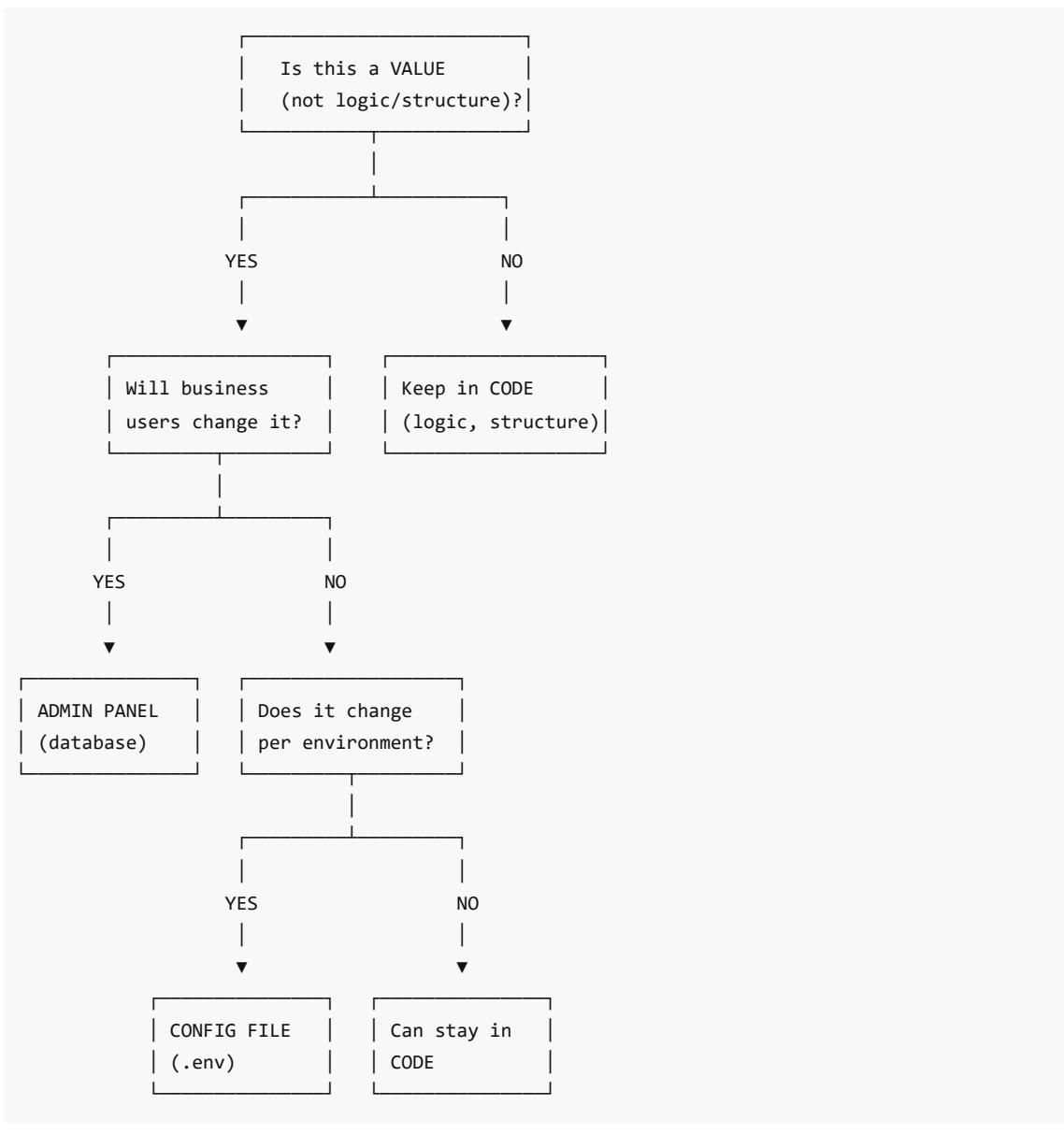
🟡 SHOULD BE IN CONFIG FILE (Not Code)

Category	Examples	Why
API Endpoints	https://api.openai.com/v1	Rarely changes, but shouldn't be in code
Timeouts	30 second API timeout	Tuning parameter
Retry Settings	3 retries, exponential backoff	Operational tuning
Log Levels	DEBUG, INFO, ERROR	Environment-specific

CAN STAY IN CODE

Category	Examples	Why
UI Layout	Grid columns, spacing	Design decisions
Component Structure	React component hierarchy	Architecture
Business Logic	Credit calculation formula	Core functionality
Validation Rules	Email format regex	Rarely changes
Error Messages	"Invalid input"	Part of UX design

Visual Decision Tree



PROMPT 1: Controlled AI Changes (Think Before Coding) {#prompt-1-controlled-changes}

The Prompt

CONTROLLED CHANGE PROTOCOL

Before making ANY code changes, you MUST follow this exact process:

PHASE 1: ANALYSIS (No Code Changes Yet)

1. ****State the Goal****: What functionality am I trying to add/fix/modify?

2. ****Impact Analysis****: List ALL files that will be affected:

- File path
- What will change in this file
- Why this file needs to change

3. ****Value Check****: Identify any values being added/modified:

- Is this a hardcoded value? (✗ Bad)
- Should this come from admin config? (✓ Good)
- Should this come from environment? (✓ Good)

4. ****Dependency Check****:

- Does frontend change require backend change?
- Does backend change require database change?
- Are there any mismatches between layers?

5. ****Show the BEFORE state****: For each file, show the relevant code BEFORE changes

PHASE 2: PROPOSAL (Still No Code Changes)

Present changes in this format:

===== PROPOSED =====

CHANGE #1 =====

 FILE: src/OneMindAI.tsx 🖥 LOCATION: Lines 250-260 ⏲ PURPOSE: Add support for GPT-5 model

BEFORE: [REDACTED] |

const models = ['gpt-4o', 'gpt-4o-mini']; |

AFTER: [REDACTED] | const

models = ['gpt-4o', 'gpt-4o-mini', 'gpt-5']; |

 WARNING: This adds a hardcoded value. RECOMMENDATION: Fetch from admin config instead.

PHASE 3: APPROVAL

Ask: "Do you approve these changes? Reply with:

- 'APPROVED' to proceed with all changes
- 'APPROVED #1, #3' to proceed with specific changes only
- 'MODIFY #2: [your instructions]' to adjust a change
- 'REJECT' to cancel all changes"

PHASE 4: IMPLEMENTATION (Only After Approval)

After receiving approval:

1. Make ONLY the approved changes
2. Show each change as it's made
3. Provide rollback instructions for each change

PHASE 5: VERIFICATION

After changes are complete:

1. List all files modified
2. Show the AFTER state of each change
3. Explain how to test the changes
4. Provide exact rollback commands if needed

RULES (Never Break These)

1. ✗ NEVER make changes without showing BEFORE/AFTER first
2. ✗ NEVER add hardcoded values for: models, pricing, tokens, limits
3. ✗ NEVER modify multiple files without listing all of them first
4. ✗ NEVER assume a change is "small enough" to skip this process
5. ✓ ALWAYS wait for explicit "APPROVED" before changing code
6. ✓ ALWAYS check if a value should come from admin config
7. ✓ ALWAYS show rollback instructions

Why This Prompt Works

Feature	Benefit
Forced Analysis Phase	AI must think before coding
BEFORE/AFTER Display	You see exactly what will change
Explicit Approval Gate	Nothing changes without your "APPROVED"
Hardcoded Value Warning	AI flags potential config issues
Rollback Instructions	Easy to undo if something breaks
Multi-File Awareness	Shows all affected files upfront

PROMPT 2: Full System Story (Code Flow Documentation) {#prompt-2-full-story}

The Prompt

FULL SYSTEM STORY REQUEST

Trace the complete data flow for: [DESCRIBE THE FEATURE/FLOW]

Required Output Format

1. ENTRY POINT

Where does this flow start? (User click, API call, scheduled job, etc.)

2. COMPLETE FLOW DIAGRAM

[USER ACTION] | ▼

[Component/Function Name] | | FILE: [exact file path] | | FUNCTION: [function name] (line XXX) | | | INPUT: | |
• param1: [type] = [example value] | | • param2: [type] = [example value] | | | PROCESS: | | 1. [What happens first] | | 2. [What happens next] | | | OUTPUT: | | • result: [type] = [example value] | | | HARDCODED VALUES USED: | | • MAX_TOKENS = 4096 (line 45) ⚠ Should be admin config | | | EXTERNAL CALLS: | | • None / API call to X / Database query Y |
| | [What data is passed] ▼ | |
STEP 2: [Next Component/Function] | | ... |

3. EXTERNAL SYSTEM INTERACTIONS

For each external system (API, database, CRM, file system):

| | | | | EXTERNAL:
[System Name] | | | DIRECTION: Outbound / Inbound / Both | | PROTOCOL: REST API / GraphQL / WebSocket / File I/O | | ENDPOINT: [URL or path] | | | DATA SENT: | | { | "model": "gpt-4o", | | "messages": [...], | | "max_tokens": 4096 | | } | | | DATA RECEIVED: | | { | "choices": [...], | | "usage": { "total_tokens": 150 } | | } | | | ERROR HANDLING: | | • 429: Retry with backoff | | • 500: Show error to user |

4. DATA TRANSFORMATION POINTS

Show where data changes format:

TRANSFORMATION: [Name] LOCATION: [file:line]

BEFORE: { userPrompt: "Hello", selectedModel: "gpt-4o" }

```
AFTER: { messages: [{ role: "user", content: "Hello" }], model: "gpt-4o" }
```

WHY: Converting UI state to API request format

5. HARDCODED VALUES INVENTORY

List ALL hardcoded values found in this flow:

Value	Location	Current Value	Should Be
MAX_TOKENS	OneMindAI.tsx:45	4096	Admin config
API_URL	proxy-client.ts:9	localhost:3002	Environment
RETRY_COUNT	error-recovery.ts:20	3	Config file

6. POTENTIAL FAILURE POINTS

Step	What Can Fail	Current Handling	Recommendation
API Call	Network timeout	Retry 3x	Add circuit breaker
DB Query	Connection lost	Crash	Add fallback

Why This Prompt Works

Feature	Benefit
Step-by-Step Trace	See exact execution order
File + Line References	Know exactly where to look
Input/Output at Each Step	See data transformation
External System Documentation	Understand integrations
Hardcoded Value Detection	Find config issues
Failure Point Analysis	Identify risks

🔍 PROMPT 3: Live Debug Output (Real-Time Execution Tracing) {#prompt-3-live-debug}

The Prompt

LIVE DEBUG IMPLEMENTATION REQUEST

Add comprehensive debug logging to [FEATURE/FLOW] that shows real-time execution in the browser console and optionally on-screen.

Requirements

1. DEBUG PANEL COMPONENT

Create a floating debug panel that shows:

- Current execution step
- Function being called
- Input parameters (with values)
- Output values
- Time taken
- Any errors

2. LOGGING FORMAT

Every significant operation should log:

```
```javascript
// Console output format
● [STEP 1/5] handleSubmit() called
 ○ File: OneMindAI.tsx:2500
 🎨 Input: { prompt: "Hello world", engines: ["gpt-4o", "claude"] }
 🕒 Started: 14:32:05.123

● [STEP 1/5] handleSubmit() completed
 🎨 Output: { requestId: "abc123", estimatedCost: 0.05 }
 🕒 Duration: 45ms

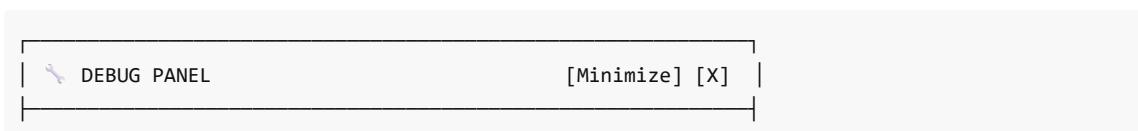
● [STEP 2/5] streamFromProvider() called
 ○ File: OneMindAI.tsx:1536
 🎨 Input: { engine: "gpt-4o", prompt: "Hello world", maxTokens: 4096 }
 🌐 External Call: POST https://api.openai.com/v1/chat/completions
 🎨 Request Body: { model: "gpt-4o", messages: [...], stream: true }

● [STEP 2/5] Streaming response...
 📈 Tokens received: 50/4096
 📈 Content length: 234 chars

● [STEP 2/5] streamFromProvider() completed
 🎨 Output: { content: "...", tokensUsed: 150 }
 🕒 Duration: 2340ms
 💰 Cost: 0.0015 credits

● [STEP 3/5] ERROR in deductCredits()
 ○ File: credit-service.ts:45
 ✖ Error: Insufficient credits
 📄 Stack: ...
```

## 3. ON-SCREEN DEBUG PANEL



```

▶ CURRENT FLOW: AI Query Execution
▶ STATUS: Running (Step 2 of 5)

STEP 1: handleSubmit ✓ (45ms)
Input: { prompt: "Hello", engines: 2 }
Output: { requestId: "abc123" }

STEP 2: streamFromProvider 🚧 (running...)
Engine: gpt-4o
Tokens: 50/4096 (1.2%)
API REQUEST
POST https://api.openai.com/v1/chat/completions
{ model: "gpt-4o", max_tokens: 4096, ... }

PENDING: deductCredits, updateUI, logAnalytics

EXTERNAL CALLS
• OpenAI API: 1 call (2.3s)
• Supabase: 0 calls
• HubSpot: 0 calls

[Export Log] [Clear] [Pause]

```

#### 4. IMPLEMENTATION APPROACH

```

// debug-logger.ts
class DebugLogger {
 private steps: DebugStep[] = [];
 private listeners: ((step: DebugStep) => void)[] = [];

 startStep(name: string, file: string, line: number, input: any) {
 const step: DebugStep = {
 id: crypto.randomUUID(),
 name,
 file,
 line,
 input,
 startTime: Date.now(),
 status: 'running'
 };
 this.steps.push(step);
 }
}
```

```

 this.emit(step);
 console.log(` [STEP ${this.steps.length}] ${name} called`);
 console.log(` 📁 File: ${file}:${line}`);
 console.log(` 📥 Input:`, input);
 return step.id;
}

endStep(stepId: string, output: any) {
 const step = this.steps.find(s => s.id === stepId);
 if (step) {
 step.output = output;
 step.endTime = Date.now();
 step.duration = step.endTime - step.startTime;
 step.status = 'completed';
 this.emit(step);
 console.log(` [STEP] ${step.name} completed`);
 console.log(` 📥 Output:`, output);
 console.log(` ⏳ Duration: ${step.duration}ms`);
 }
}

logExternalCall(type: string, url: string, request: any, response: any) {
 console.log(`🌐 External Call: ${type} ${url}`);
 console.log(` 📬 Request:`, request);
 console.log(` 📮 Response:`, response);
}

logError(stepId: string, error: Error) {
 const step = this.steps.find(s => s.id === stepId);
 if (step) {
 step.error = error;
 step.status = 'error';
 this.emit(step);
 console.log(`🔴 ERROR in ${step.name}`);
 console.log(` ✗ ${error.message}`);
 console.log(` 📋 Stack:`, error.stack);
 }
}

export const debugLogger = new DebugLogger();

```

## 5. USAGE IN CODE

```

// In OneMindAI.tsx
async function handleSubmit() {
 const stepId = debugLogger.startStep(
 'handleSubmit',
 'OneMindAI.tsx',
 2500,
 { prompt, selectedEngines: engines.filter(e => e.selected).map(e => e.id) }
)
}

```

```

);
try {
 // ... existing code ...

 debugLogger.endStep(stepId, { requestId, estimatedCost });

} catch (error) {
 debugLogger.LogError(stepId, error);
 throw error;
}
}
}

```

## 6. TOGGLE DEBUG MODE

```

// Enable via URL param: ?debug=true
// Or via keyboard shortcut: Ctrl+Shift+D
// Or via admin panel setting

```

## Why This Prompt Works

Feature	Benefit
<b>Visual Debug Panel</b>	See execution in real-time without console
<b>Step-by-Step Logging</b>	Understand exact execution order
<b>Input/Output Visibility</b>	See data at every step
<b>External Call Tracking</b>	Know what's sent to APIs
<b>Error Highlighting</b>	Quickly spot failures
<b>Exportable Logs</b>	Share debug info for troubleshooting

## IMPLEMENTATION CHECKLIST {#implementation-checklist}

### Phase 1: Database Setup (Day 1)

- Create `ai_models` table in Supabase
- Create `system_config` table in Supabase
- Migrate existing hardcoded models to database
- Migrate existing hardcoded pricing to database
- Add RLS policies for admin-only write access

### Phase 2: Admin Panel (Day 2-3)

- Create Models management page

- Create System Config management page
- Add validation for all config inputs
- Add audit logging for config changes
- Test real-time updates

### Phase 3: Frontend Integration (Day 4)

- Create `useAdminConfig` hook
- Replace hardcoded models with hook data
- Replace hardcoded pricing with hook data
- Add loading states for config fetch
- Add fallback for config fetch failure

### Phase 4: Debug System (Day 5-6)

- Create `DebugLogger` class
- Create `DebugPanel` component
- Add debug logging to key functions
- Add toggle mechanism (URL param, keyboard shortcut)
- Test with real user flows

### Phase 5: Documentation (Day 7)

- Document all admin-configurable values
  - Document debug panel usage
  - Add comments to prevent AI from hardcoding
  - Create runbook for common config changes
- 

## APPENDIX: Quick Reference

### Values That MUST Be Admin-Controlled

- ✓ AI model list (names, IDs, versions)
- ✓ Model pricing (input/output per million tokens)
- ✓ Token limits (per model)
- ✓ Rate limits (requests per minute)
- ✓ Feature flags (enable/disable providers)
- ✓ User roles and prompts
- ✓ Signup bonus credits
- ✓ Markup percentage

### Values That Can Stay in Code

- ✓ UI component structure
- ✓ CSS/styling
- ✓ Validation logic

- ✓ Error message templates
- ✓ Business logic formulas
- ✓ React component hierarchy

## Values That Should Be in .env

- ✓ API keys (never in code or database)
- ✓ Database connection strings
- ✓ Third-party service URLs
- ✓ Environment-specific settings (dev/staging/prod)

*Document generated for OneMindAI project*

*Purpose: Establishing controlled AI-assisted development practices*