Leadership Platform - AI Code Review Package

Project Overview

A Next.js 14 + Supabase educational platform for high school leadership development featuring:
- **Student Dashboard**: Interactive learning with Socratic AI conversations
- **Teacher Dashboard**: Class management, student progress tracking, content review
- **3D World Building**: Students earn and place items in their virtual world
- **Gamified Learning**: 45-lesson curriculum across 4 phases

Live URL

https://leadership-platform-flame.vercel.app/

Tech Stack

- **Frontend**: Next.js 14, React 18, TypeScript, Tailwind CSS, Framer Motion
- **Backend**: Supabase (PostgreSQL, Auth, RLS, Edge Functions)
- **3D Graphics**: Three.js / React-Three-Fiber
- **AI**: Anthropic Claude API (via Edge Functions)

REVIEW PROMPT FOR ChatGPT/Gemini

Copy this prompt along with the code files below:

PROMPT:

You are a senior software architect conducting a comprehensive code review for an educational platform. Please analyze the following codebase and provide:

1. FUNCTIONALITY VERIFICATION

- Does the student login flow work correctly?
- Is the task completion tracking (Do Now, Scenario, Challenge) properly implemented?
- Are Supabase queries efficient and correct?
- Is data properly persisted after user actions?

2. SECURITY AUDIT

- Are there any SQL injection vulnerabilities?
- Is Row Level Security (RLS) properly configured?
- Are API keys properly protected?
- Is authentication handled securely?
- Are there any XSS vulnerabilities?
- Is user input properly sanitized?

3. PERFORMANCE CONCERNS

- Are there unnecessary re-renders?
- Are database queries optimized?
- Is code splitting implemented correctly?
- Are there memory leaks in useEffect hooks?

4. CODE QUALITY

- Are TypeScript types properly used?
- Is error handling comprehensive?
- Are edge cases handled?
- Is the code maintainable and well-structured?

5. FEATURE SUGGESTIONS

Based on the educational context (high school leadership), suggest:
- Additional gamification features
- Enhanced AI interaction patterns
- Improved teacher analytics
- Student engagement features

6. CRITICAL ISSUES

List any bugs or issues that could cause the application to fail or behave unexpectedly.

KEY CODE FILES

File 1: Student Dashboard (src/app/student/page.tsx)

```
'use client'

import React, { useState } from 'react'
import { motion, AnimatePresence } from 'framer-motion'
import { LogOut, Wifi, WifiOff, Clock, Sparkles, Home, Users, Star,
Trophy, Lightbulb } from 'lucide-react'
import dynamic from 'next/dynamic'
import { useStudentData } from './hooks/useStudentData'
import type { StudentTab } from './types'
import HomeTab from './tabs/HomeTab'
import WorldTab from './tabs/WorldTab'
import CommonsTab from './tabs/CommonsTab'
import DiscoverTab from './tabs/DiscoverTab'
import ProgressTab from './tabs/ProgressTab'

const SocraticModal = dynamic(() => import('@/components/
SocraticModal'), { ssr: false })
const ChallengeSubmissionModal = dynamic(() => import('@/components/
ChallengeSubmissionModal'), { ssr: false })
const GatewayChallenge = dynamic(() => import('@/components/
GatewayChallenge'), { ssr: false })
const DailyUnmask = dynamic(() => import('@/components/DailyUnmask'),
{ ssr: false })
const BrainstormAI = dynamic(() => import('@/components/
BrainstormAI'), { ssr: false })

export default function StudentDashboard() {
  const {
    user, lesson, progress, world, helpRequests, teacherChallenges,
    discoveries, journalData, connectionMapData, placedItems,
inventoryItems,
    discoveredSecrets, gatewayComplete, dailyUnmaskData, studentSkills,
    loading, isOnline, error, loadData, handleLogout,
refreshHelpRequests,
    markTextAnchorComplete, markMediaComplete, updateWorld,
updateProgress
  } = useStudentData()

  const [activeTab, setActiveTab] = useState<StudentTab>('home')
  const [showWelcome, setShowWelcome] = useState(false)
  const [showDoNow, setShowDoNow] = useState(false)
  const [showExitTicket, setShowExitTicket] = useState(false)
```

```
  const [showScenario, setShowScenario] = useState(false)
  const [showChallenge, setShowChallenge] = useState(false)
  const [showGateway, setShowGateway] = useState(false)
  const [showBrainstorm, setShowBrainstorm] = useState(false)
  const [showDailyUnmask, setShowDailyUnmask] = useState(false)

  React.useEffect(() => {
    const shouldShowWelcome =
localStorage.getItem('leadership_show_welcome')
    if (shouldShowWelcome === 'true' && user?.name) {
      setShowWelcome(true)
      localStorage.removeItem('leadership_show_welcome')
    }
  }, [user])

  const addToInventory = async (itemType: string) => {
    if (!user?.id) return
    try {
      const res = await fetch('/api/inventory', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          studentId: user.id,
          itemType,
          earnedFrom: 'lesson_completion',
          earnedDescription: `Completed ${lesson?.skill_name ||
'activity'}`
        })
      })
      if (res.ok) loadData()
    } catch (e) {
      console.error('Failed to add inventory:', e)
    }
  }

  const onDoNowComplete = async () => {
    updateProgress({ do_now_complete: true, status: 'in_progress' })
    updateWorld({ flowers: (world?.flowers || 0) + 1 })
    await addToInventory('flower')
    setShowDoNow(false)
  }
```

```
  const onScenarioComplete = async () => {
    updateProgress({ scenario_complete: true })
    updateWorld({ trees: (world?.trees || 0) + 1 })
    await addToInventory('tree')
    setShowScenario(false)
    setShowChallenge(true)
  }

  if (loading) return <LoadingState />
  if (error) return <ErrorState error={error} onRetry={loadData} />

  return (
    <div className="min-h-screen bg-surface-950">
      {/* Header with connection status */}
      <header className="sticky top-0 z-40 glass-card">
        <div className="max-w-6xl mx-auto px-4 py-3 flex items-center
justify-between">
          <div className="flex items-center gap-3">
            <motion.div className="text-3xl" animate={{ rotate: [0,
10, -10, 0] }}>🌱</motion.div>
            <div>
              <h1 className="font-bold text-surface-100">Leadership
2.0</h1>
              <p className="text-xs text-surface-500">Phase
{lesson?.phase_id}</p>
            </div>
          </div>
          <div className={`flex items-center gap-1.5 px-2.5 py-1.5
rounded-lg text-xs ${
            isOnline ? 'bg-green-500/20 text-green-400' : 'bg-
red-500/20 text-red-400'
          }`}>
            {isOnline ? <Wifi size={14} /> : <WifiOff size={14} />}
          </div>
        </div>
      </header>

      {/* Welcome Popup */}
      <AnimatePresence>
        {showWelcome && (
          <motion.div className="fixed inset-0 z-50 flex items-center
justify-center bg-black/60">
```

```
            <motion.div className="bg-surface-800 rounded-2xl p-8 text-
center">
              <div className="text-6xl mb-4">👋</div>
              <h2 className="text-2xl font-bold text-white mb-2">Hello
{user?.name}!</h2>
              <p className="text-surface-400 mb-6">Welcome back to
your leadership journey!</p>
              <button onClick={() => setShowWelcome(false)}
className="btn-primary">
                Let's Go!
              </button>
            </motion.div>
          </motion.div>
        )}
      </AnimatePresence>

      {/* Tab Navigation & Content */}
      <main className="max-w-6xl mx-auto p-4">
        <TabNavigation activeTab={activeTab}
setActiveTab={setActiveTab} />
        {activeTab === 'home' && (
          <HomeTab
            lesson={lesson} progress={progress}
            onDoNow={() => setShowDoNow(true)}
            onScenario={() => setShowScenario(true)}
            onChallenge={() => setShowChallenge(true)}
          />
        )}
        {activeTab === 'world' && <WorldTab />}
        {activeTab === 'commons' && <CommonsTab />}
        {activeTab === 'discover' && <DiscoverTab />}
        {activeTab === 'progress' && <ProgressTab />}
      </main>

      {/* Modals */}
      {showDoNow && (
        <SocraticModal
          type="do_now"
          lesson={lesson}
          userId={user?.id}
          classId={user?.class_id}
          userName={user?.name}
```

```
          onClose={() => setShowDoNow(false)}
          onComplete={onDoNowComplete}
        />
      )}
      {showScenario && (
        <SocraticModal
          type="scenario"
          lesson={lesson}
          userId={user?.id}
          classId={user?.class_id}
          userName={user?.name}
          onClose={() => setShowScenario(false)}
          onScenarioComplete={onScenarioComplete}
        />
      )}
    </div>
  )
}
```

File 2: Socratic AI Modal (src/components/SocraticModal.tsx)

```
'use client'

import { useState, useRef, useEffect } from 'react'
import { motion } from 'framer-motion'
import { X, Send, Check } from 'lucide-react'
import { createClient } from '@/lib/supabase/client'

interface SocraticModalProps {
  type: 'do_now' | 'scenario' | 'exit_ticket'
  lesson: LessonData | null
  userId: string | undefined
  classId: string | undefined
  userName: string | undefined
  onClose: () => void
  onComplete?: () => void
  onScenarioComplete?: (challengePlan: string) => void
}

export default function SocraticModal({
  type, lesson, userId, classId, userName, onClose, onComplete,
onScenarioComplete
}: SocraticModalProps) {
  const [messages, setMessages] = useState<Array<{ role: 'assistant' |
'user', content: string }>>([])
  const [input, setInput] = useState('')
  const [loading, setLoading] = useState(false)
  const [complete, setComplete] = useState(false)
  const [responseCount, setResponseCount] = useState(0)
  const supabase = createClient()

  const MIN_RESPONSES = 5 // Minimum exchanges required

  useEffect(() => {
    let greeting = ''
    if (type === 'do_now') {
      greeting = `Hey <span class="math-inline" style="display:
inline;"><math xmlns="http://www.w3.org/1998/Math/MathML"
display="inline"><mrow><mrow><mi>u</mi><mi>s</mi><mi>e</mi><mi>r</
mi><mi>N</mi><mi>a</mi><mi>m</mi><mi>e</mi><mo
stretchy="false">&#x0007C;</mo><msup><mo stretchy="false">&#x0007C;</
mo><mi>&#x02032;</mi></msup><mi>t</mi><mi>h</mi><mi>e</mi><mi>r</
mi><msup><mi>e</mi><mi>&#x02032;</mi></msup></mrow><mo>&#x00021;</
```

```
mo><mi>R</mi><mi>e</mi><mi>a</mi><mi>d</mi><mi>y</mi><mi>t</mi><mi>o</
mi><mi>w</mi><mi>a</mi><mi>r</mi><mi>m</mi><mi>u</mi><mi>p</mi><mi>t</
mi><mi>h</mi><mi>a</mi><mi>t</mi><mi>b</mi><mi>r</mi><mi>a</mi><mi>i</
mi><mi>n</mi><mo>&#x0003F;</mo><mi>\n</mi><mi>\n</mi><mo>&#x0002A;</
mo><mo>&#x0002A;</mo><mi>"</mi></mrow></math></
span>{lesson?.compelling_question}"**\n\nWhat comes to mind first?`
    } else if (type === 'scenario') {
      greeting = `Time to put $
{lesson?.skill_name} into practice. I'll push back and make you think.
Ready?`
    } else {
      greeting = `Let's wrap up today's learning about $
{lesson?.skill_name}. What stuck with you?`
    }
    setMessages([{ role: 'assistant', content: greeting }])
  }, [type, lesson])

  const sendMessage = async () => {
    if (!input.trim() || loading) return

    // Enforce minimum word count for quality responses
    if (input.trim().split(' ').length < 3 && responseCount <
MIN_RESPONSES) {
      setMessages(prev => [...prev,
        { role: 'user', content: input.trim() },
        { role: 'assistant', content: "Give me at least a full
sentence. What are you really thinking?" }
      ])
      setInput('')
      return
    }

    const userMessage = input.trim()
    setInput('')
    setMessages(prev => [...prev, { role: 'user', content:
userMessage }])
    setLoading(true)
    setResponseCount(prev => prev + 1)

    try {
      const response = await fetch('/api/ai/socratic', {
        method: 'POST',
```

```
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          type, lesson_id: lesson?.id, student_id: userId, class_id:
classId,
          conversation_history: messages, user_message: userMessage,
          lesson_context: lesson, response_count: responseCount + 1,
min_responses: MIN_RESPONSES
        })
      })

      const data = await response.json()

      if (data.crisis_detected) {
        setMessages(prev => [...prev, { role: 'assistant', content:
data.message }])
        return
      }

      setMessages(prev => [...prev, { role: 'assistant', content:
data.message }])

      if (data.should_complete && responseCount >= MIN_RESPONSES - 1) {
        setComplete(true)

        // Save session to appropriate table
        const tableName = type === 'do_now' ? 'do_now_sessions' :
                          type === 'exit_ticket' ?
'exit_ticket_sessions' : 'scenario_sessions'

        await supabase.from(tableName).insert({
          student_id: userId, lesson_id: lesson?.id, class_id: classId,
          conversation: [...messages, { role: 'user', content:
userMessage }, { role: 'assistant', content: data.message }],
          response_count: responseCount + 1,
          completed_at: new Date().toISOString(),
          ...(type !== 'scenario' ? { date: new
Date().toISOString().split('T')[0] } : {})
        })
      }
    } catch (error) {
      console.error('Error:', error)
      setMessages(prev => [...prev, { role: 'assistant', content:
```

```
'Connection issue. Try again.' }])
    } finally {
      setLoading(false)
    }
  }

  return (
    <motion.div className="fixed inset-0 z-50 flex items-center
justify-center bg-black/60 p-4">
      <div className="bg-surface-800 rounded-2xl w-full max-w-2xl max-
h-[85vh] flex flex-col">
        <header className="flex items-center justify-between p-4
border-b border-surface-700">
          <h2 className="font-bold text-lg">
            {type === 'do_now' ? 'Do Now' : type === 'scenario' ?
'Real-Life Scenario' : 'Exit Ticket'}
          </h2>
          <button onClick={onClose}><X /></button>
        </header>

        <div className="flex-1 overflow-y-auto p-4 space-y-4">
          {messages.map((msg, i) => (
            <div key={i} className={`flex ${msg.role === 'user' ?
'justify-end' : 'justify-start'}`}>
              <div className={`max-w-[80%] rounded-xl p-3 ${
                msg.role === 'user' ? 'bg-primary-600 text-white' :
'bg-surface-700 text-surface-100'
              }`}>
                {msg.content}
              </div>
            </div>
          ))}
        </div>

        {!complete ? (
          <div className="p-4 border-t border-surface-700 flex gap-2">
            <input
              value={input}
              onChange={(e) => setInput(e.target.value)}
              onKeyDown={(e) => e.key === 'Enter' && sendMessage()}
              placeholder="Type your response..."
              className="flex-1 bg-surface-700 rounded-xl px-4 py-2"
```

```
            />
            <button onClick={sendMessage} disabled={loading}
className="btn-primary">
              <Send size={18} />
            </button>
          </div>
        ) : (
          <div className="p-4 text-center">
            <button onClick={() => onComplete?.()} className="btn-
primary">
              <Check size={18} /> Complete & Earn Reward
            </button>
          </div>
        )}
      </div>
    </motion.div>
  )
}
```

File 3: Teacher Dashboard (src/app/teacher/page.tsx) - Key Sections

```
'use client'

import { useEffect, useState } from 'react'
import { createClient } from '@/lib/supabase/client'
import { useRouter } from 'next/navigation'

export default function TeacherDashboard() {
  const [user, setUser] = useState<any>(null)
  const [classes, setClasses] = useState<ClassData[]>([])
  const [selectedClass, setSelectedClass] = useState<ClassData |
null>(null)
  const [students, setStudents] = useState<StudentData[]>([])
  const [pendingReviews, setPendingReviews] =
useState<SubmissionData[]>([])
  const [crisisAlerts, setCrisisAlerts] = useState<CrisisAlert[]>([])
  const [activeTab, setActiveTab] = useState<'overview' | 'reviews' |
'students' | 'curriculum' | 'donow' | 'conversations' | 'worlds' |
'metrics' | 'alerts' | 'settings'>('overview')

  const router = useRouter()
  const supabase = createClient()

  useEffect(() => {
    loadInitialData()
  }, [])

  const loadInitialData = async () => {
    const { data: { user: authUser } } = await supabase.auth.getUser()
    if (!authUser) { router.push('/'); return }

    const { data: userData } = await supabase
      .from('users').select('*').eq('id', authUser.id).single()

    if (!userData || userData.role !== 'teacher') { router.push('/');
return }
    setUser(userData)

    const { data: classesData } = await supabase
      .from('classes').select('*').eq('teacher_id',
authUser.id).order('name')

    if (classesData?.length) {
```

```
      setClasses(classesData)
      setSelectedClass(classesData[0])
    }
  }

  // 10 Tabs: overview, reviews, students, curriculum, donow,
conversations, worlds, metrics, alerts, settings
  // Each tab fetches and displays relevant Supabase data

  return (
    <div className="min-h-screen bg-surface-950">
      <header>{/* Teacher header with class selector */}</header>
      <nav>{/* Tab navigation for 10 dashboard sections */}</nav>
      <main>
        {activeTab === 'overview' && <OverviewSection />}
        {activeTab === 'curriculum' && <CurriculumSection />}
        {activeTab === 'conversations' && <AIConversationsSection />}
        {activeTab === 'worlds' && <StudentWorldsSection />}
        {/* ... other tabs */}
      </main>
    </div>
  )
}
```

DATABASE SCHEMA (Key Tables)

```
-- Users (students and teachers)
CREATE TABLE users (
  id UUID PRIMARY KEY REFERENCES auth.users,
  email TEXT UNIQUE,
  name TEXT,
  role TEXT CHECK (role IN ('student', 'teacher')),
  class_id UUID REFERENCES classes(id),
  password_hash TEXT,
  created_at TIMESTAMPTZ DEFAULT NOW()
);

-- Classes
CREATE TABLE classes (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  name TEXT NOT NULL,
  code TEXT UNIQUE NOT NULL,
  teacher_id UUID REFERENCES users(id),
  current_lesson_id INTEGER DEFAULT 1,
  created_at TIMESTAMPTZ DEFAULT NOW()
);

-- Lessons (45 total curriculum)
CREATE TABLE lessons (
  id INTEGER PRIMARY KEY,
  phase_id INTEGER,
  class_number INTEGER,
  skill_name TEXT,
  compelling_question TEXT,
  lesson_objective TEXT,
  text_anchor_title TEXT,
  media_url TEXT
);

-- Do Now Sessions (Socratic conversations)
CREATE TABLE do_now_sessions (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  student_id UUID REFERENCES users(id),
  lesson_id INTEGER REFERENCES lessons(id),
  class_id UUID REFERENCES classes(id),
  conversation JSONB,
  response_count INTEGER,
  date DATE,
```

```
  completed_at TIMESTAMPTZ
);

-- Student Progress
CREATE TABLE student_lessons (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  student_id UUID REFERENCES users(id),
  lesson_id INTEGER REFERENCES lessons(id),
  do_now_complete BOOLEAN DEFAULT FALSE,
  scenario_complete BOOLEAN DEFAULT FALSE,
  challenge_complete BOOLEAN DEFAULT FALSE,
  status TEXT DEFAULT 'not_started'
);

-- Student Inventory (earned items)
CREATE TABLE student_inventory (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  student_id UUID REFERENCES users(id),
  item_type TEXT,
  earned_from TEXT,
  created_at TIMESTAMPTZ DEFAULT NOW()
);

-- World Placements (3D world building)
CREATE TABLE world_placements (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  student_id UUID REFERENCES users(id),
  item_type TEXT,
  x FLOAT, y FLOAT, z FLOAT,
  created_at TIMESTAMPTZ DEFAULT NOW()
);
```

RLS POLICIES

```
-- Students can only see/modify their own data
CREATE POLICY "Students own data" ON users FOR ALL USING (auth.uid() =
id);

-- Teachers can insert their own classes
CREATE POLICY "Teachers insert classes" ON classes FOR INSERT WITH
CHECK (auth.uid() = teacher_id);

-- Teachers can read students in their classes
CREATE POLICY "Teachers read students" ON users FOR SELECT
USING (role = 'student' AND class_id IN (SELECT id FROM classes WHERE
teacher_id = auth.uid()));
```

QUESTIONS FOR REVIEWER

1. Is the minimum response count (5) for Socratic conversations appropriate for educational engagement?
2. Are there any race conditions in the inventory/progress update flow?
3. Is the crisis detection in AI responses adequate for student safety?
4. Should we add more granular RLS policies for multi-tenant isolation?
5. What additional analytics would help teachers understand student engagement?

Please provide a detailed assessment covering all sections above.