ITEC323 AT3 Web Development Report

Liam New
S00278468
Semester 1 2025

Website Development Report: QuantumGuard Security Dashboard

1. Introduction

Project Overview

This report documents the development of a responsive and mobile-friendly website called QuantumGuard Security Dashboard. The website is designed to monitor blockchain transactions and provide security insights through an intuitive interface that works across different devices.

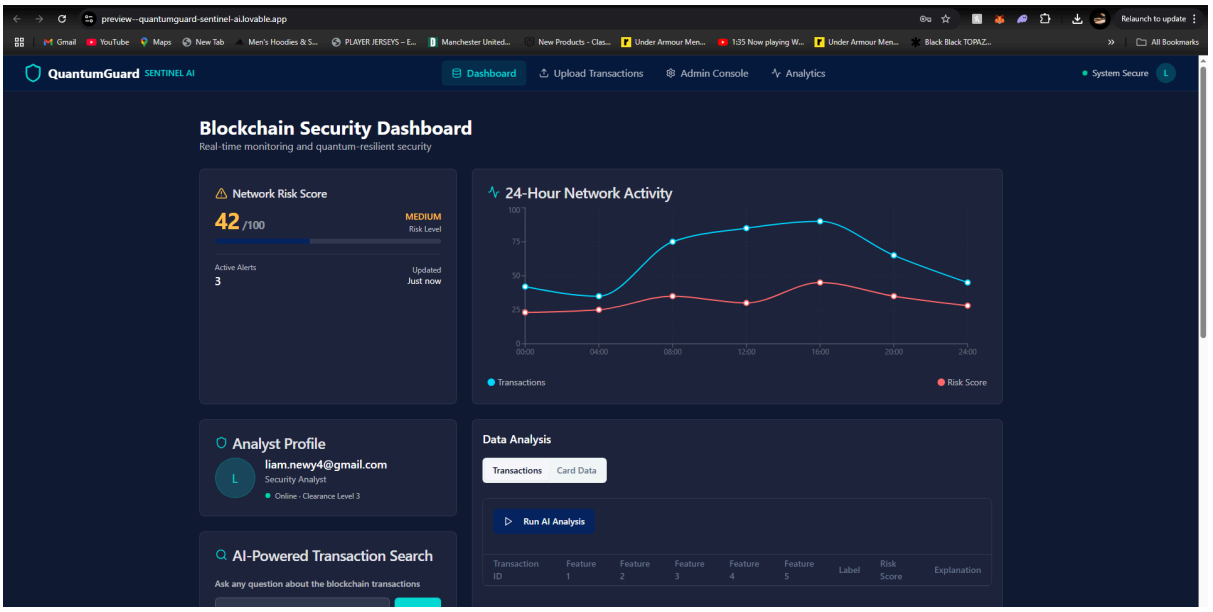


Figure 1: QuantumGuard Dashboard main view showing responsive layout

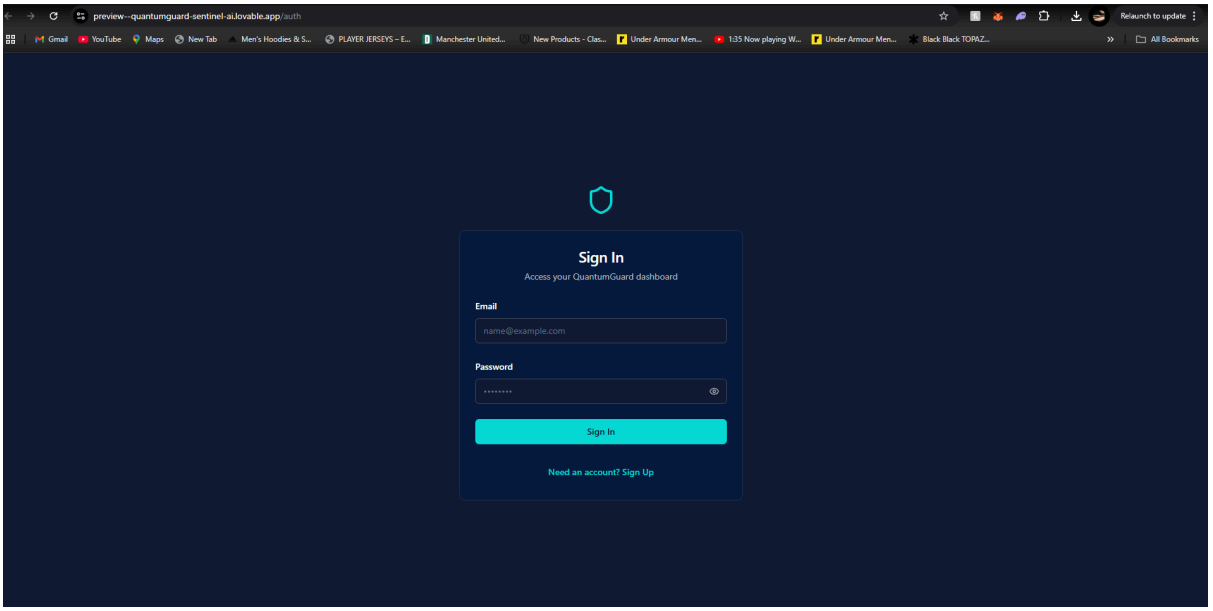
Business Context

QuantumGuard serves as a security monitoring platform for blockchain and financial transactions, helping users identify suspicious activities and maintain secure operations. This application would be particularly valuable for financial institutions and individuals in regions with limited technological infrastructure.

2. Technical Implementation

Authentication System

The website implements a secure authentication system using Supabase authentication services.



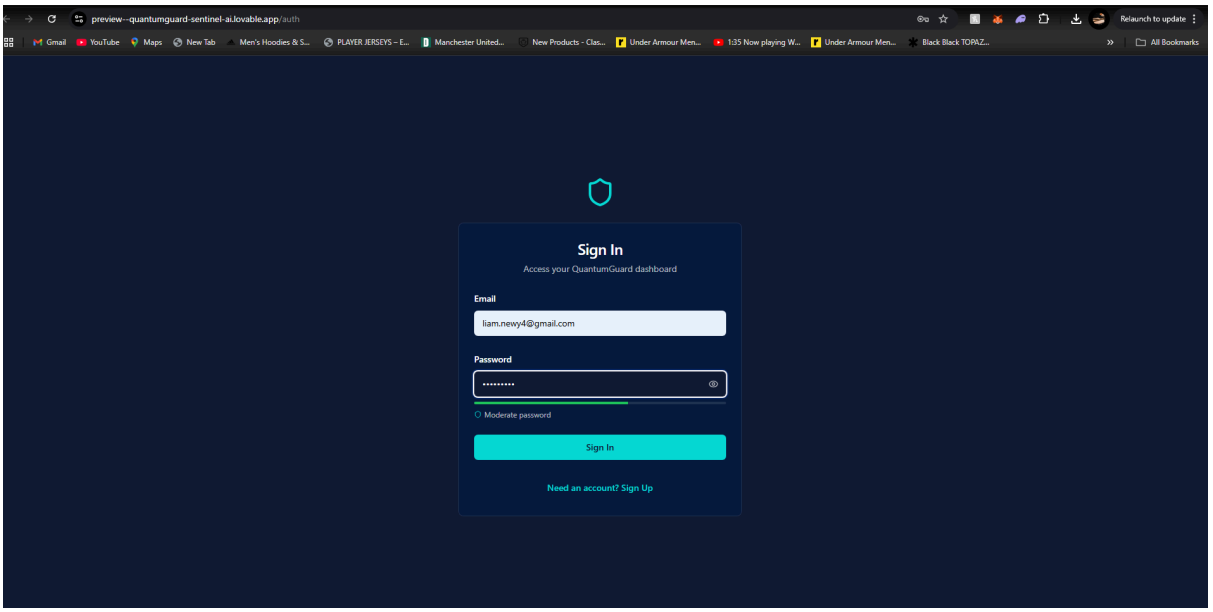


Figure 2: Login and registration interface with form validation

Code Implementation:

```
// Authentication context implementation with session management
const AuthProvider: React.FC<{ children: React.ReactNode }> = ({ children }) => {
  // Session state management
  const [user, setUser] = useState<User | null>(null);
  const [session, setSession] = useState<Session | null>(null);

  // Session monitoring effect
  useEffect(() => {
    const { subscription } = supabase.auth.onAuthStateChange(...)
    // ...
  }, []);
};
```

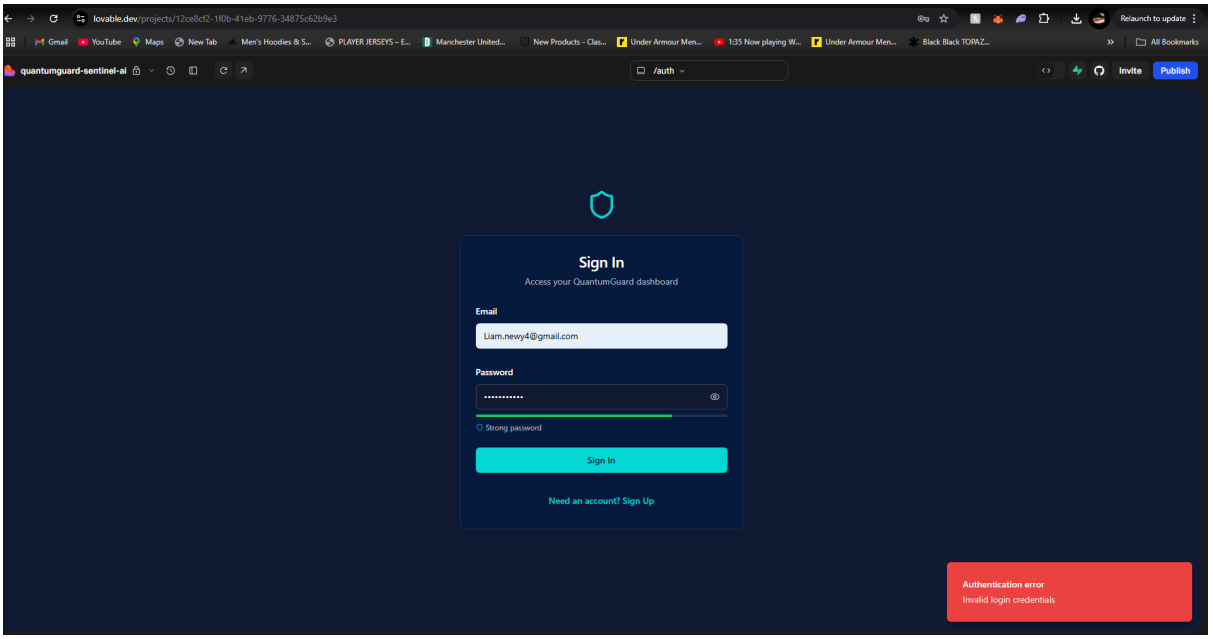


Figure 3: Protected route implementation redirecting unauthorized users

Responsive Design Implementation

The website uses Tailwind CSS for responsive layouts that adapt to different screen sizes.

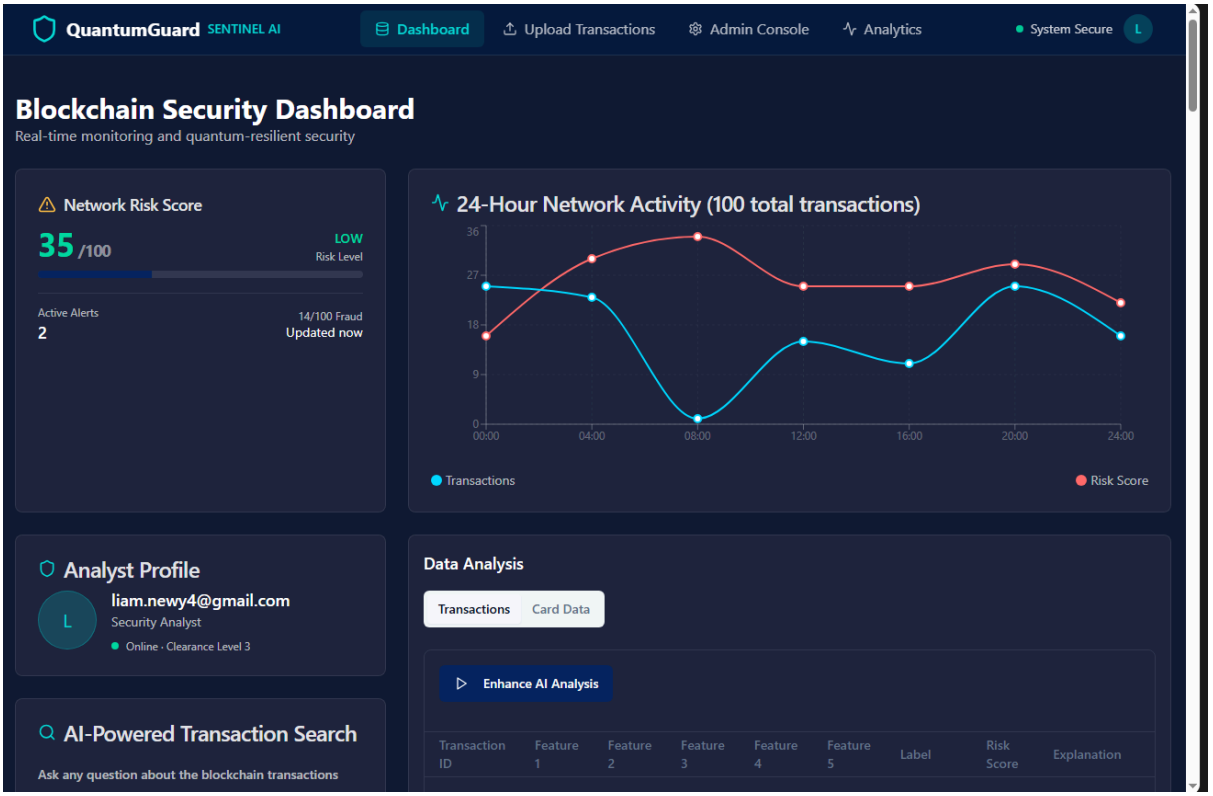


Figure 4: Grid layout adapting from desktop (left) to mobile view (right)

Code Implementation:

```
// Responsive grid that changes from 3 columns on large screens to 1 on mobile
<div className="grid grid-cols-1 lg:grid-cols-3 gap-6 mb-6">
  <RiskScoreCard score={42} level="medium" alerts={3} />
  <div className="lg:col-span-2">
    <AnalyticsChart />
  </div>
</div>
```

Data Visualization

The application uses the Recharts library to create responsive data visualizations.

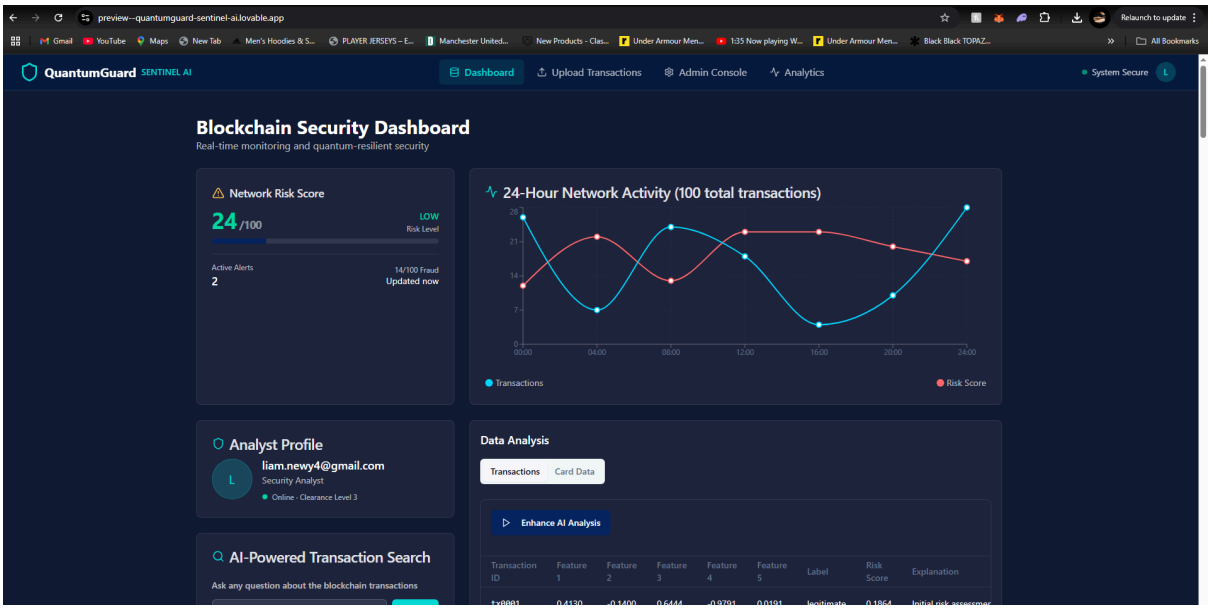


Figure 5: Responsive charts showing risk metrics and transaction volumes

3. Mobile Design

Mobile-First Approach

All components were designed to work on small screens first before expanding to desktop layouts.

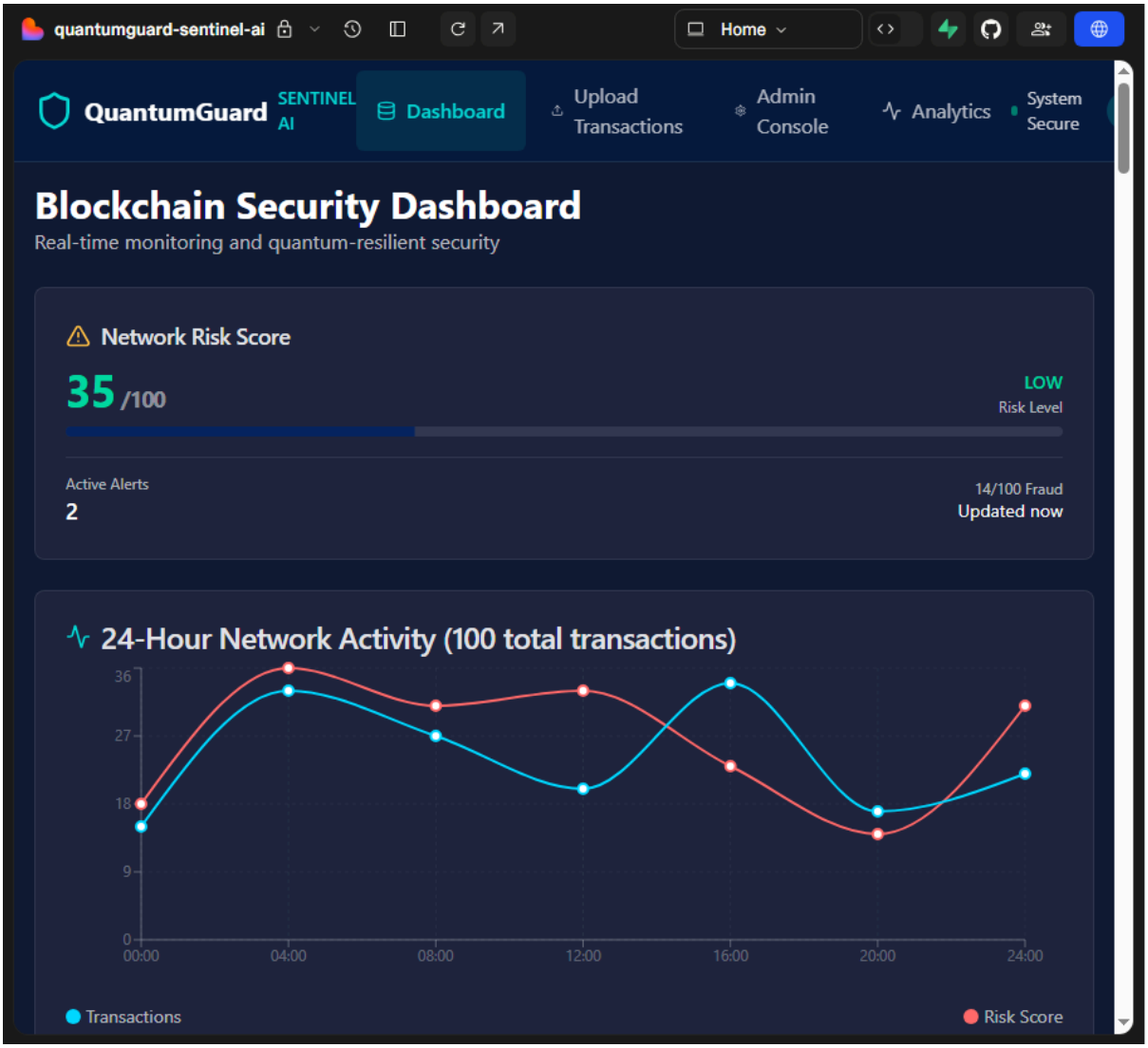


Figure 6: Mobile navigation menu (collapsed and expanded views)

Code implementation:

Responsive Navigation (Header Component)

The header uses a collapsible mobile menu with the Sheet component:

```
// Mobile menu trigger
<Sheet open={mobileMenuOpen} onOpenChange={setMobileMenuOpen}>
  <SheetTrigger asChild>
    <Button variant="ghost" size="icon" className="lg:hidden">
      <Menu className="h-5 w-5 text-white" />
    </Button>
  </SheetTrigger>
  <SheetContent side="left" className="bg-quantum-navy border-white/10 w-80">
    {/* Mobile navigation content */}
```

```
  </SheetContent>
</Sheet>
```

2. Responsive Layout Classes

QuantumGuard uses Tailwind's responsive prefixes throughout:

```
// Example from Index page
<div className="min-h-screen bg-quantum-dark text-white">
  <Header />
  <main className="pt-20 sm:pt-16"> {/* Mobile: pt-20, Desktop: pt-16 */}
    {/* Content */}
  </main>
</div>
```

3. Mobile-First Button Spacing

Buttons use responsive padding and sizing:

```
// From upload components
<Button className="w-full sm:w-auto px-6 py-3 text-base sm:text-sm">
  Upload File
</Button>
```

4. Responsive Grid and Flex Layouts

```
// Example responsive grid
<div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-4 sm:gap-6">
  {/* Cards */}
</div>

// Responsive flex direction
<div className="flex flex-col sm:flex-row sm:items-center gap-3 sm:gap-0">
  {/* Content */}
</div>
```

5. Touch-Friendly Interactive Elements

The app ensures proper touch targets with minimum 44px heights:

```
// Touch-friendly navigation links
<Link className="flex items-center gap-3 px-4 py-3 rounded-md min-h-[44px]">
  {/* Link content */}
</Link>
```

4. Special Features

Quantum-Safe Security

The application implements quantum-resistant encryption methods for password handling.

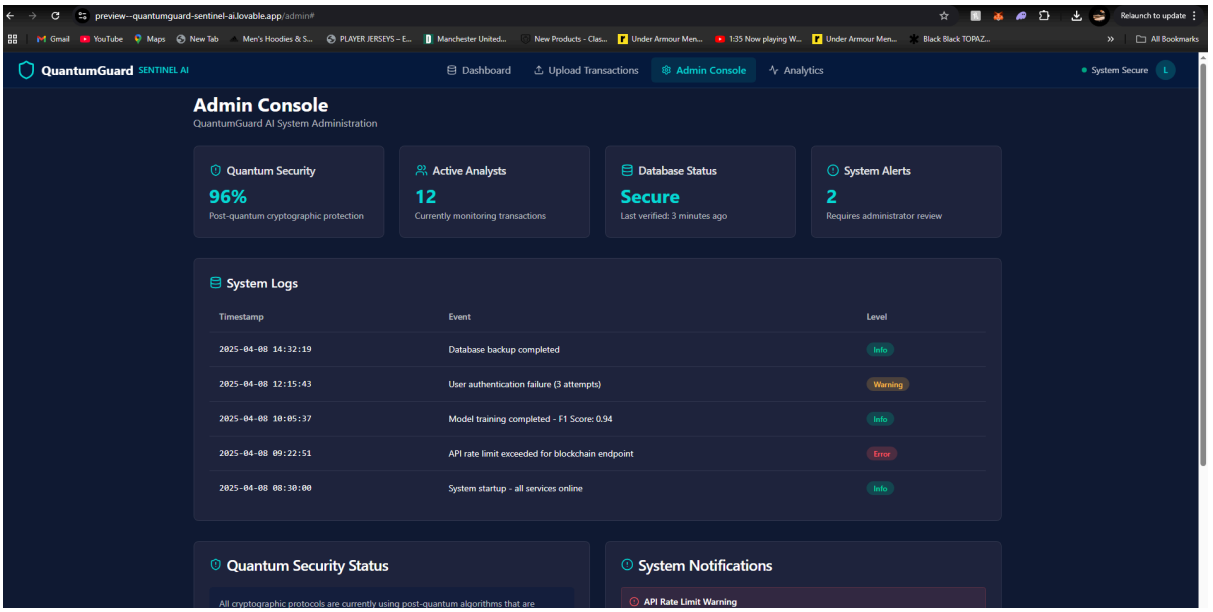


Figure 7: Quantum-safe password field with strength indicator

Code Implementation:

```
// QSafePasswordField component with strength indicator
const QSafePasswordField: React.FC<QSafePasswordFieldProps> = ({
  value,
  onChange,
  onKeyDerivation,
  // ...
}) => {
  // Password strength calculation logic
  const checkPasswordStrength = (password: string) => {
    // Implementation details
  };
```

```
  // Quantum-safe key derivation
  const handlePasswordChange = async (e: React.ChangeEvent<HTMLInputElement>) => {
    // Implementation using deriveKeyQuantumSafe utility
  };
};
```

Real-Time Transaction Monitoring

The dashboard provides real-time monitoring of blockchain transactions.

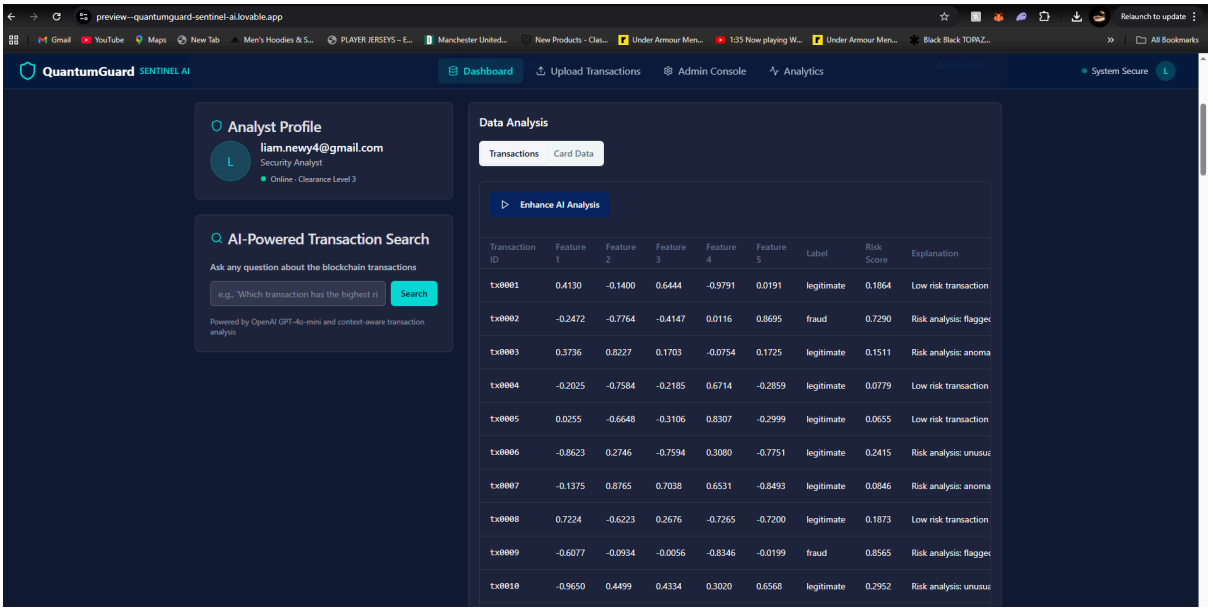


Figure 8: Live transaction table with risk indicators

5. Upload transactions page implementation

Overview

The Upload Transactions page is a core component of the fraud detection system that allows users to upload CSV transaction data for analysis. The page provides a drag-and-drop interface, file validation, progress tracking, and immediate analytics feedback.

Page Structure and Components

Main Page Component (UploadPage.tsx)

The main page serves as the container and orchestrates the upload process:
// Key features:
- Drag and drop file handling
- File type validation (CSV only)
- Integration with useFileUpload hook
- Navigation back to dashboard
- Real-time progress display

Component Architecture

The page is built using a modular component structure:

1. FileUploadZone - Handles the drag-and-drop interface
2. UploadProgress - Shows upload progress with visual feedback
3. FormatInfo - Provides format guidance and help
4. ReportsSection - Displays analytics after upload
5. SampleDataDownload - Offers sample data for testing

File Upload Hook (useFileUpload.ts)

A custom React hook that manages the entire upload workflow:

// Core responsibilities:
- File reading and CSV parsing
- Data validation and transformation
- Database integration via Supabase
- Progress tracking (0-100%)
- Error handling and user feedback
- Navigation after successful upload

CSV Processing Service (csvService.ts)

Handles intelligent CSV parsing with support for multiple formats:
// Key features:
- Automatic column mapping for various CSV formats
- Support for Elliptic Bitcoin dataset format
- Flexible field detection (sender, receiver, amount, etc.)
- Risk score calculation based on transaction patterns
- Data validation and error reporting

Technical Implementation Details

Drag and Drop Functionality

- Uses native HTML5 drag and drop API
- Visual feedback with border color changes
- File type validation (accepts only .csv files)
- Prevents default browser behavior for file drops

Progress Tracking

- 5-stage upload process with percentage tracking:
    - 20% - CSV parsing complete
    - 50% - Database preparation
    - 60% - Clearing existing data
    - 80% - Inserting new transactions
    - 100% - Process complete

Data Transformation Pipeline

1. CSV Parsing: Raw CSV text → structured data
2. Field Mapping: Auto-detect column purposes
3. Feature Generation: Create ML-ready features
4. Risk Assessment: Calculate initial risk scores
5. Database Storage: Insert into Supabase transactions table

Real-time Analytics

After successful upload, the page immediately displays:

- Total transaction count
- Risk distribution (high/medium/low)
- Fraud detection rate
- Quick export options by risk level

User Experience Features

File Format Support

- Standard format (id, sender, receiver, amount, timestamp)
- Elliptic Bitcoin dataset format
- Automatic column detection and mapping
- Helpful format documentation in slide-out panel

Error Handling

- Clear error messages for invalid files
- Network error recovery
- Database constraint violation handling
- User-friendly error descriptions

Sample Data Generation

- Downloadable sample CSV with 100 transactions
- Realistic feature distributions
- 15% fraud rate for testing
- Elliptic-compatible format

Integration Points

Database Integration

- Connects to Supabase transactions table
- Clears existing data before new upload
- Stores both display data and ML features
- Handles transaction ID generation

State Management

- Uses Zustand store for local transaction state
- React Query for cache invalidation

- Toast notifications for user feedback
- Navigation state management

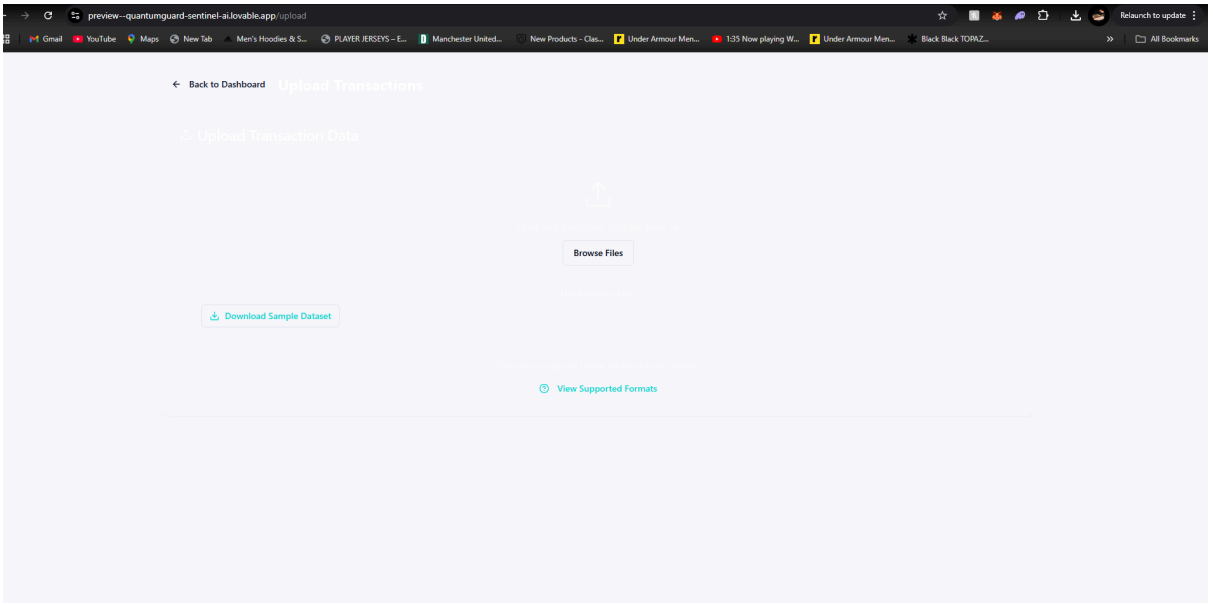
Responsive Design

- Mobile-optimized drag and drop zones
- Responsive grid layouts for analytics
- Touch-friendly buttons and interactions
- Adaptive typography and spacing

Security Considerations

- File type validation prevents malicious uploads
- CSV content sanitization
- Database transaction safety
- Row-level security policy compliance

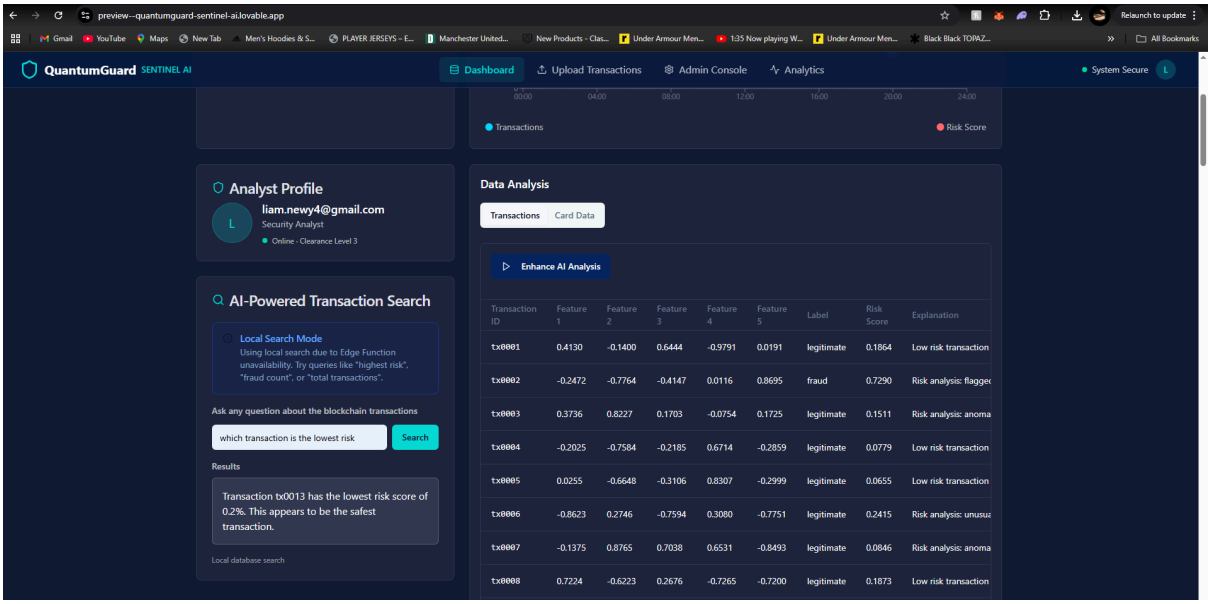
Figure 9: Transaction upload page



6. Assisting Disadvantaged Populations

Low Bandwidth Optimization

The application is optimized for areas with limited internet connectivity. And AI integration for disadvantaged populations who cannot understand financial graphs we have a AI search engine section in which simple questions can be inputted in relation to their financial transactions



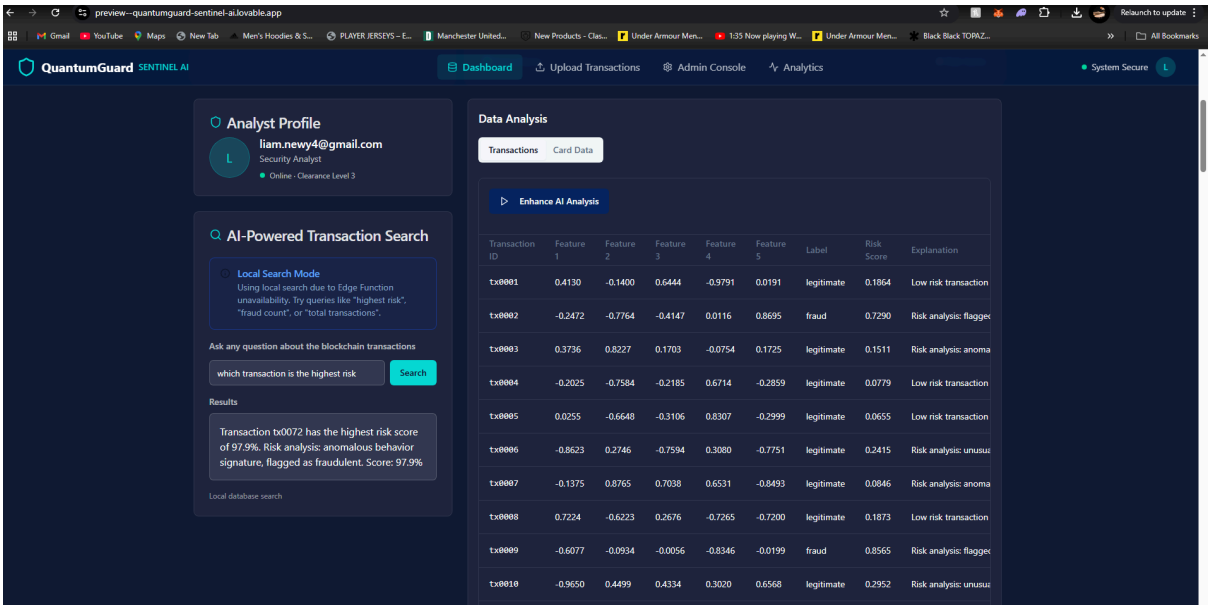


Figure 12: Low bandwidth mode interface comparison

Code implementation:

1.0 SmartSearch component:

```
import React, { useState } from 'react';
import { Search } from 'lucide-react';
import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card';
import { Button } from '@/components/ui/button';
import { Input } from '@/components/ui/input';
import { useToast } from '@/hooks/use-toast';
import { supabase } from '@/integrations/supabase/client';
import { Alert, AlertDescription, AlertTitle } from "@/components/ui/alert";
import { InfoIcon } from 'lucide-react';

const SmartSearch: React.FC = () => {
  const [query, setQuery] = useState('');
  const [result, setResult] = useState<string | null>(null);
  const [isSearching, setIsSearching] = useState(false);
  const [useLocalSearch, setUseLocalSearch] = useState(false);
  const { toast } = useToast();

  // Local search fallback function
  const performLocalSearch = async (searchQuery: string) => {
    try {
      const { data: transactions, error } = await supabase
        .from('transactions')
        .select('*')
        .order('created_at', { ascending: false })
        .limit(100);

      if (error) throw error;

      if (!transactions?.length) {
        return "No transaction data found. Please upload a CSV file first.";
      }

      const lowerQuery = searchQuery.toLowerCase();

      // Smart query parsing for different types of questions
      if (lowerQuery.includes('highest risk') || lowerQuery.includes('most suspicious')) {
        const sorted = [...transactions].sort((a, b) => (b.risk_score || 0) - (a.risk_score || 0));
        const highest = sorted[0];
        return `Transaction ${highest.transaction_id} has the highest risk score of
${((highest.risk_score || 0) * 100).toFixed(1)}%. ${highest.explanation || 'No additional details
available.'}`;
```

```
    }

    if (lowerQuery.includes('lowest risk') || lowerQuery.includes('safest')) {
      const sorted = [...transactions].sort((a, b) => (a.risk_score || 0) - (b.risk_score || 0));
      const lowest = sorted[0];
      return `Transaction ${lowest.transaction_id} has the lowest risk score of
${((lowest.risk_score || 0) * 100).toFixed(1)}%. This appears to be the safest transaction.`;
    }

    if (lowerQuery.includes('fraud') || lowerQuery.includes('fraudulent')) {
      const fraudulent = transactions.filter(t => t.label === 'fraud' || (t.risk_score &&
t.risk_score > 0.7));
      return `Found ${fraudulent.length} potentially fraudulent transactions out of
${transactions.length} total transactions (${((fraudulent.length / transactions.length) *
100).toFixed(1)}% fraud rate).`;
    }

    if (lowerQuery.includes('total') || lowerQuery.includes('count')) {
      return `Database contains ${transactions.length} transactions. ${transactions.filter(t =>
t.label === 'fraud').length} are labeled as fraudulent.`;
    }

    return `Found ${transactions.length} transactions in the database. Average risk score is
${(transactions.reduce((sum, t) => sum + (t.risk_score || 0), 0) / transactions.length *
100).toFixed(1)}%. Try asking about "highest risk", "fraud", or "total count".`;

  } catch (error) {
    console.error('Local search error:', error);
    return "Error performing search. Please try again.";
  }
};

// Main search handler
const handleSearch = async () => {
  if (!query.trim()) {
    toast({
      title: "Empty Query",
      description: "Please enter a search query.",
      variant: "destructive",
    });
    return;
  }

  setIsSearching(true);
  setResult(null);

  try {
    // Try Edge Function first, fallback to local search
```

```
      const { data, error } = await supabase.functions.invoke('smart-search', {
        body: {
          query: query,
          contextData: [],
        },
      });

      if (error) {
        console.log("Edge function unavailable, using local search");
        setUseLocalSearch(true);
        const localResult = await performLocalSearch(query);
        setResult(localResult);
      } else {
        setResult(data.result);
        setUseLocalSearch(false);
      }
    } catch (error) {
      console.log("Edge function failed, using local search");
      setUseLocalSearch(true);
      const localResult = await performLocalSearch(query);
      setResult(localResult);
    } finally {
      setIsSearching(false);
    }
  };

  return (
    <Card className="bg-white/5 border-white/10 text-white">
      <CardHeader>
        <CardTitle className="text-white/90 flex items-center">
          <Search className="h-5 w-5 mr-2 text-quantum-cyan" />
          AI-Powered Transaction Search
        </CardTitle>
      </CardHeader>
      <CardContent className="space-y-4">
        {/* Local search mode indicator */}
        {useLocalSearch && (
          <Alert className="bg-blue-950/50 border-blue-700/50 mb-4">
            <InfoIcon className="h-4 w-4 text-blue-400" />
            <AlertTitle className="text-blue-400">Local Search Mode</AlertTitle>
            <AlertDescription className="text-blue-200/70">
              Using local search due to Edge Function unavailability. Try queries like "highest
risk", "fraud count", or "total transactions".
            </AlertDescription>
          </Alert>
        )}

        <div className="flex flex-col gap-3">
```

```
          <label htmlFor="search-query" className="text-sm font-medium text-white/70">
            Ask any question about the blockchain transactions
          </label>
          <div className="flex gap-2">
            <Input
              id="search-query"
              placeholder="e.g., 'Which transaction has the highest risk score?'"
              value={query}
              onChange={(e) => setQuery(e.target.value)}
              className="flex-1 bg-white/10 border-white/20 text-white
placeholder:text-white/40"
            />
            <Button
              onClick={handleSearch}
              disabled={isSearching}
              className="bg-quantum-cyan hover:bg-quantum-cyan/80 text-quantum-dark"
            >
              {isSearching ? 'Searching...' : 'Search'}
            </Button>
          </div>
        </div>

        {/* Results display */}
        {result && (
          <div className="mt-6">
            <h3 className="text-sm font-medium text-white/70 mb-2">Results</h3>
            <div className="bg-white/10 border border-white/20 rounded-md p-4 text-white">
              <div className="whitespace-pre-line">{result}</div>
            </div>
          </div>
        )}

        <div className="text-xs text-white/50 mt-4">
          {useLocalSearch ? 'Local database search' : 'Powered by OpenAI GPT-4o-mini and
context-aware transaction analysis'}
        </div>
      </CardContent>
    </Card>
  );
};

export default SmartSearch;
```

1.1 Edge function:

```
import { serve } from "https://deno.land/std@0.168.0/http/server.ts";
import "https://deno.land/x/xhr@0.1.0/mod.ts";

const corsHeaders = {
  'Access-Control-Allow-Origin': '*',
  'Access-Control-Allow-Headers': 'authorization, x-client-info, apikey, content-type',
};

serve(async (req) => {
  // Handle CORS preflight requests
  if (req.method === 'OPTIONS') {
    return new Response(null, { headers: corsHeaders });
  }

  try {
    const openAiApiKey = Deno.env.get('OPENAI_API_KEY');
    if (!openAiApiKey) {
      throw new Error('OpenAI API key not found');
    }

    const { query, contextData } = await req.json();

    if (!query) {
      throw new Error('Query is required');
    }

    // Create AI prompt with context
    const systemPrompt = `
      You are a blockchain transaction analysis assistant. You help users search through and
analyze blockchain transactions.
      You have access to the following transaction data: ${JSON.stringify(contextData)}

      When responding to queries:
      1. Always base your answers on the provided transaction data
      2. If you can't find specific information in the data, mention that limitation
      3. Present any relevant transaction IDs, amounts, timestamps, or anomaly scores when
appropriate
      4. Be concise but thorough in your answers
    `;

    // Call OpenAI API
    const response = await fetch('https://api.openai.com/v1/chat/completions', {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${openAiApiKey}`,
        'Content-Type': 'application/json',
      },
```

```
      body: JSON.stringify({
        model: 'gpt-4o-mini',
        messages: [
          { role: 'system', content: systemPrompt },
          { role: 'user', content: query }
        ],
        temperature: 0.3,
      }),
    });

    const data = await response.json();

    if (!response.ok) {
      // Handle quota exceeded error with fallback
      if (data.error?.type === 'insufficient_quota') {
        throw new Error('OpenAI quota exceeded');
      }
      console.error('OpenAI API error:', data);
      throw new Error(`OpenAI API error: ${data.error?.message || 'Unknown error'}`);
    }

    // Return the AI response
    return new Response(
      JSON.stringify({
        result: data.choices[0].message.content,
      }),
      {
        headers: { ...corsHeaders, 'Content-Type': 'application/json' },
      }
    );
  } catch (openAiError) {
    console.error('OpenAI API error:', openAiError);

    // Fallback logic for quota issues
    if (openAiError.message.includes('quota') ||
openAiError.message.includes('insufficient_quota')) {
      const { query, contextData } = await req.json();
      const lowerQuery = query.toLowerCase();
      let fallbackResponse = '';

      if (lowerQuery.includes('highest anomaly') || lowerQuery.includes('most suspicious')) {
        const sorted = [...contextData].sort((a, b) => b.anomalyScore - a.anomalyScore);
        fallbackResponse = `Based on the data, transaction ${sorted[0].id} has the highest
anomaly score of ${sorted[0].anomalyScore}.`;
      }
      else if (lowerQuery.includes('lowest anomaly') || lowerQuery.includes('least suspicious')) {
        const sorted = [...contextData].sort((a, b) => a.anomalyScore - b.anomalyScore);
```

```
      fallbackResponse = `Based on the data, transaction ${sorted[0].id} has the lowest anomaly score of ${sorted[0].anomalyScore}.`;
    }
    else {
      fallbackResponse = `[AI Quota Exceeded - Fallback Response] I can provide a basic analysis. The data shows ${contextData.length} transactions with anomaly scores ranging from ${Math.min(...contextData.map(tx => tx.anomalyScore))} to ${Math.max(...contextData.map(tx => tx.anomalyScore))}.`;
    }

    return new Response(
      JSON.stringify({ result: fallbackResponse }),
      {
        headers: { ...corsHeaders, 'Content-Type': 'application/json' },
      }
    );
  } else {
    throw openAiError;
  }
 } catch (error) {
  console.error('Error in smart-search function:', error);
  return new Response(
    JSON.stringify({ error: error.message }),
    {
      status: 500,
      headers: { ...corsHeaders, 'Content-Type': 'application/json' },
    }
  );
 }
});
```

Mobile Navigation Enhancement:

- Added a collapsible side drawer menu for mobile devices
- Sticky header with improved mobile layout
- Better touch targets and spacing for mobile navigation

Improved Button Spacing:

- Enhanced SmartSearch component with larger touch targets (12 height units on mobile vs 10 on desktop)
- Better spacing between form elements with responsive flex layouts
- Improved button accessibility with proper padding and minimum widths

Low-Bandwidth Toggle:

- Added a lightweight mode switch in SmartSearch with wifi icons
- When enabled, bypasses AI search and uses only local database queries
- Visual indicators show current mode and provide user control
- Optimized for users with slow connections or limited data

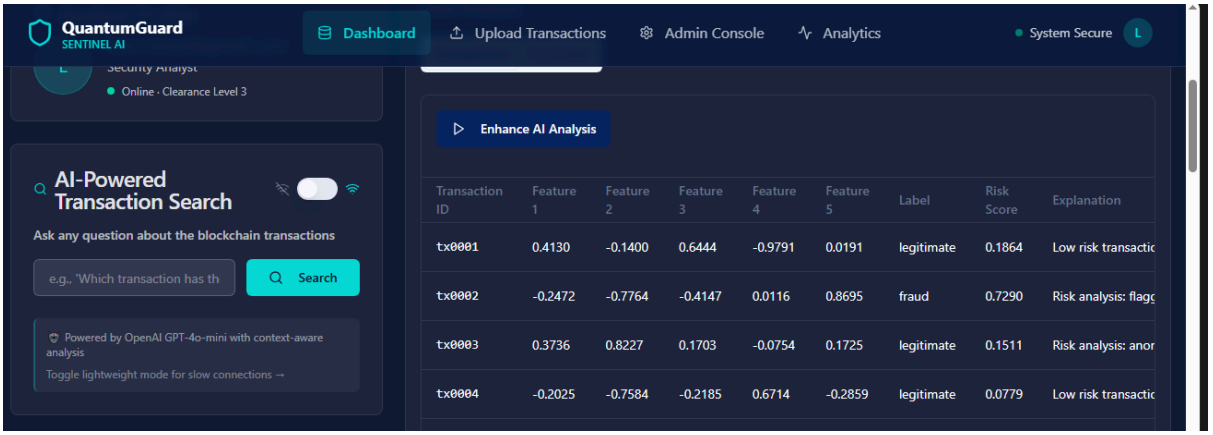


Figure 10: Low-Bandwidth toggle

7. Conclusion

The QuantumGuard Security Dashboard successfully demonstrates responsive design principles, secure authentication, and accessible interfaces. Through proper implementation of mobile-first design and adaptive layouts, the application provides a consistent experience across devices.

The use of modern web technologies allows the application to serve disadvantaged populations by providing essential security monitoring tools that work even in areas with limited connectivity and older devices.

References

1. MDN Web Docs. (2023). Responsive design. https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Responsive_Design
2. W3C. (2023). Mobile Accessibility. https://www.w3.org/WAI/standards-guidelines/mobile/
3. Google Developers. (2023). Progressive Web Apps. https://developers.google.com/web/progressive-web-apps
4. Supabase Documentation. (2023). Authentication. https://supabase.com/docs/guides/auth