Neon Alarm HTML Source Code

Generated by Gemini

November 28, 2025

File: index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Neon Alarm</title>
    <!-- Tailwind CSS -->
    <script src="https://cdn.tailwindcss.com"></script>
    <!-- Google Fonts: Share Tech Mono for that digital look -->
    <link href="https://fonts.googleapis.com/css2?family=Share+Tech+Mono&display=swap" rel="stylesheet">
    <!-- FontAwesome -->
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.0/css/all.min.css">

    <style>
        :root {
            --neon-green: #39ff14;
            --dark-bg: #050505;
            --glass-bg: rgba(20, 20, 20, 0.8);
        }

        body {
            background-color: var(--dark-bg);
            color: var(--neon-green);
            font-family: 'Share Tech Mono', monospace;
            overflow-x: hidden;
        }

        /* Neon Glow Effects */
        .neon-text {
            text-shadow: 0 0 5px rgba(57, 255, 20, 0.5),
                         0 0 10px rgba(57, 255, 20, 0.3);
        }

        .neon-text-intense {
            text-shadow: 0 0 10px var(--neon-green),
                         0 0 20px var(--neon-green),
                         0 0 40px var(--neon-green);
        }

        .neon-border {
            box-shadow: 0 0 5px var(--neon-green), inset 0 0 5px var(--neon-green);
            border: 1px solid var(--neon-green);
        }

        .neon-box-shadow {
            box-shadow: 0 0 15px rgba(57, 255, 20, 0.2);
        }

        /* Custom Scrollbar */
        ::-webkit-scrollbar {
            width: 8px;
        }
        ::-webkit-scrollbar-track {
            background: #000;
        }
        ::-webkit-scrollbar-thumb {
```

```
            background: #1a4d10;
            border-radius: 4px;
        }
        ::-webkit-scrollbar-thumb:hover {
            background: var(--neon-green);
        }

        /* Animations */
        @keyframes pulse-red {
            0% { box-shadow: 0 0 0 0 rgba(255, 50, 50, 0.7); }
            70% { box-shadow: 0 0 0 20px rgba(255, 50, 50, 0); }
            100% { box-shadow: 0 0 0 0 rgba(255, 50, 50, 0); }
        }

        @keyframes shake {
            0% { transform: translate(1px, 1px) rotate(0deg); }
            10% { transform: translate(-1px, -2px) rotate(-1deg); }
            20% { transform: translate(-3px, 0px) rotate(1deg); }
            30% { transform: translate(3px, 2px) rotate(0deg); }
            40% { transform: translate(1px, -1px) rotate(1deg); }
            50% { transform: translate(-1px, 2px) rotate(-1deg); }
            60% { transform: translate(-3px, 1px) rotate(0deg); }
            70% { transform: translate(3px, 1px) rotate(-1deg); }
            80% { transform: translate(-1px, -1px) rotate(1deg); }
            90% { transform: translate(1px, 2px) rotate(0deg); }
            100% { transform: translate(1px, -2px) rotate(-1deg); }
        }

        .alarm-ringing {
            animation: shake 0.5s cubic-bezier(.36,.07,.19,.97) both infinite;
            border-color: #ff3939 !important;
            color: #ff3939 !important;
            text-shadow: 0 0 10px #ff3939 !important;
            box-shadow: 0 0 20px #ff3939, inset 0 0 10px #ff3939 !important;
        }

        .stop-btn-pulse {
            animation: pulse-red 1.5s infinite;
        }

        /* Toggle Switch */
        .toggle-checkbox:checked {
            right: 0;
            border-color: var(--neon-green);
        }
        .toggle-checkbox:checked + .toggle-label {
            background-color: var(--neon-green);
        }

        /* Time Input Styling */
        input[type="time"]::-webkit-calendar-picker-indicator {
            filter: invert(1) sepia(1) saturate(5) hue-rotate(90deg);
            cursor: pointer;
        }

        /* Tone Selection */
        .tone-btn.active {
            background-color: var(--neon-green);
            color: black;
            box-shadow: 0 0 10px var(--neon-green);
        }

        /* Loading Spinner */
        .spinner {
            border: 4px solid rgba(255, 255, 255, 0.3);
            border-top: 4px solid var(--neon-green);
            border-radius: 50%;
            width: 24px;
            height: 24px;
            animation: spin 1s linear infinite;
        }

        @keyframes spin {
```

```
            0% { transform: rotate(0deg); }
            100% { transform: rotate(360deg); }
        }
    </style>
</head>
<body class="flex flex-col h-screen w-full items-center justify-center relative select-none">

    <!-- Modal for Ringing Alarm & Focus Prompt -->
    <div id="alarmModal" class="fixed inset-0 z-50 flex items-center justify-center bg-black bg-opacity-95 hidden transition-opacity duration-300">
        <div class="alarm-ringing bg-black border-2 border-red-500 p-8 rounded-lg flex flex-col items-center gap-6 max-w-sm w-full mx-4 relative overflow-hidden transition-all duration-500">
            <!-- Canvas Visualizer -->
            <canvas id="audioVisualizer" class="absolute inset-0 w-full h-full opacity-30 pointer-events-none"></canvas>

            <div id="ringingState" class="relative z-10 flex flex-col items-center w-full">
                <i class="fas fa-bell fa-4x mb-2 animate-bounce"></i>
                <h2 class="text-4xl font-bold tracking-wider">WAKE UP</h2>
                <div id="ringingTime" class="text-2xl">00:00</div>
                <button onclick="stopAlarm()" class="stop-btn-pulse mt-4 px-8 py-3 bg-red-600 text-black font-bold text-xl rounded hover:bg-red-500 transition-colors uppercase tracking-widest w-full border-none outline-none">
                    DISMISS
                </button>
            </div>

            <!-- Morning Focus State (Initially Hidden) -->
            <div id="focusState" class="relative z-10 flex flex-col items-center w-full hidden">
                <h2 class="text-3xl font-bold tracking-wider mb-4 neon-text">FOCUS PROTOCOL ACTIVE</h2>
                <div id="alarmGoalDisplay" class="text-lg opacity-70 mb-4 text-center"></div>

                <button id="generateFocusBtn" onclick="initiateFocusGeneration()" class="w-full mt-2 px-8 py-3 bg-gray-800 text-white font-bold text-lg rounded neon-border hover:bg-gray-700 transition-colors uppercase tracking-widest border-none outline-none flex items-center justify-center">
                    <i class="fas fa-terminal mr-2"></i>
                    GENERATE MORNING FOCUS
                </button>

                <div id="focusResult" class="mt-4 p-4 w-full bg-black/50 rounded text-sm text-left border border-gray-700 hidden">
                    <!-- LLM generated content goes here -->
                </div>

                <div id="focusLoading" class="hidden mt-4 flex items-center text-sm text-gray-400">
                    <div class="spinner mr-2"></div> Processing request...
                </div>
            </div>
        </div>
    </div>

    <!-- Main Container -->
    <div class="w-full max-w-md h-full flex flex-col p-6 relative">

        <!-- Header / Clock -->
        <div class="flex-none pt-4 pb-4 flex flex-col items-center justify-center space-y-2">
            <div id="currentDate" class="text-sm tracking-[0.2em] opacity-80 uppercase">LOADING DATE...</div>
            <div id="clockDisplay" class="text-7xl md:text-8xl font-bold neon-text-intense tracking-tighter tabular-nums leading-none">00:00</div>
            <div id="secondsDisplay" class="text-xl tracking-widest opacity-90 font-bold self-end pr-4 -mt-2">:00</div>
        </div>

        <!-- Tone Selector -->
        <div class="flex-none mb-6 w-full">
             <div class="flex justify-between items-center mb-2">
                <span class="text-xs tracking-widest opacity-70">SYNTH PATCH</span>
                <button onclick="previewTone()" class="text-xs border border-[#39ff14] px-2 py-1 rounded hover:bg-[#39ff14] hover:text-black transition-colors"><i class="fas fa-play mr-1"></i>TEST</button>
            </div>
            <!-- Updated Tone Selector Grid to include 5 options -->
            <div class="grid grid-cols-5 gap-1">
```

```
                <button onclick="setTone('pulse')" id="btn-pulse" class="tone-btn active border border-[#39ff14] p-2 rounded text-xs font-bold tracking-wider hover:bg-[#39ff14] hover:text-black transition-all">PULSE</button>
                <button onclick="setTone('raid')" id="btn-raid" class="tone-btn border border-[#39ff14] p-2 rounded text-xs font-bold tracking-wider hover:bg-[#39ff14] hover:text-black transition-all">RAID</button>
                <button onclick="setTone('warp')" id="btn-warp" class="tone-btn border border-[#39ff14] p-2 rounded text-xs font-bold tracking-wider hover:bg-[#39ff14] hover:text-black transition-all">WARP</button>
                <button onclick="setTone('chime')" id="btn-chime" class="tone-btn border border-[#39ff14] p-2 rounded text-xs font-bold tracking-wider hover:bg-[#39ff14] hover:text-black transition-all">CHIME</button>
                <button onclick="setTone('synth')" id="btn-synth" class="tone-btn border border-[#39ff14] p-2 rounded text-xs font-bold tracking-wider hover:bg-[#39ff14] hover:text-black transition-all">SYNTH</button>
            </div>
        </div>

        <!-- Add Alarm Section -->
        <div class="flex-none mb-4 w-full">
            <!-- Time Input -->
            <div class="border border-[#39ff14] p-1 rounded-t-lg flex items-center bg-black shadow-[0_0_10px_rgba(57,255,20,0.1)]">
                <input type="time" id="alarmTimeInput" class="bg-transparent text-[#39ff14] text-xl px-4 py-2 w-full focus:outline-none font-bold text-center" required>
                <button onclick="addAlarm()" class="bg-[#39ff14] text-black hover:bg-white hover:text-black transition-all duration-200 rounded-full w-12 h-12 flex items-center justify-center flex-shrink-0 font-bold text-xl">
                    <i class="fas fa-plus"></i>
                </button>
            </div>
            <!-- Goal Input -->
            <input type="text" id="alarmGoalInput" placeholder="Optional: Enter today's primary goal (e.g., Finish project report)" class="w-full p-2 text-sm bg-black/50 border-x border-b border-[#39ff14] rounded-b-lg text-gray-300 placeholder-gray-600 focus:outline-none focus:border-white">

            <div class="text-center mt-2 text-xs text-gray-500 tracking-wider">SET TIME AND OPTIONAL GOAL</div>
        </div>

        <!-- User Name Divider (KRISH) - New Placement -->
        <div class="flex-none text-center text-3xl font-bold tracking-widest mb-6 neon-text-intense uppercase py-2">
            KRISH
        </div>

        <!-- Alarms List -->
        <div class="flex-1 overflow-y-auto pr-2 relative" id="alarmsList">
            <!-- Empty State is now handled dynamically in JS -->
        </div>

        <!-- Footer -->
        <div class="flex-none pt-4 text-center text-[10px] opacity-40 uppercase tracking-[0.3em]">
            SYSTEM READY // V.3.0.0
        </div>
    </div>

    <script>
        // --- API Configuration ---
        const apiKey = ""; // API Key is empty as per instructions
        const apiUrl = `https://generativelanguage.googleapis.com/v1beta/models/gemini-2.5-flash-preview-09-2025:generateContent?key=${apiKey}`;

        // --- State Management ---
        let alarms = JSON.parse(localStorage.getItem('neonAlarms')) || [];
        let currentTone = localStorage.getItem('neonTone') || 'pulse';
        let activeAlarm = null; // Store the alarm object that is currently ringing/dismissed

        let audioContext = null;
        let oscillator = null;
        let gainNode = null;
        let analyser = null;
        let isRinging = false;
        let intervalId = null;
        let animationFrameId = null;

        // Initialize UI State for Tone
        document.querySelectorAll('.tone-btn').forEach(btn => btn.classList.remove('active'));
```

```
        if (document.getElementById(`btn-${currentTone}`)) {
             document.getElementById(`btn-${currentTone}`).classList.add('active');
        }

        // --- Clock Logic ---
        function updateClock() {
            const now = new Date();
            const hours = String(now.getHours()).padStart(2, '0');
            const minutes = String(now.getMinutes()).padStart(2, '0');
            const seconds = String(now.getSeconds()).padStart(2, '0');

            // Date formatting
            const dateOptions = { weekday: 'short', month: 'short', day: 'numeric' };
            const dateString = now.toLocaleDateString('en-US', dateOptions).toUpperCase();

            document.getElementById('clockDisplay').textContent = `${hours}:${minutes}`;
            document.getElementById('secondsDisplay').textContent = `:${seconds}`;
            document.getElementById('currentDate').textContent = dateString;

            checkAlarms(hours, minutes, seconds);
        }

        setInterval(updateClock, 1000);
        updateClock(); // Initial call

        // --- Tone Selection ---
        function setTone(tone) {
            currentTone = tone;
            localStorage.setItem('neonTone', tone);

            // Update UI
            document.querySelectorAll('.tone-btn').forEach(btn => btn.classList.remove('active'));
            document.getElementById(`btn-${tone}`).classList.add('active');
        }

        function previewTone() {
            if(isRinging) return;
            initAudio();
            playSingleToneCycle();
        }

        // --- Alarm CRUD Logic ---

        function saveAlarms() {
            localStorage.setItem('neonAlarms', JSON.stringify(alarms));
            renderAlarms();
        }

        function addAlarm() {
            const timeInput = document.getElementById('alarmTimeInput');
            const goalInput = document.getElementById('alarmGoalInput');
            const timeValue = timeInput.value;
            const goalValue = goalInput.value.trim(); // Trim goal input

            if (!timeValue) return;

            // Prevent duplicates
            if (alarms.some(a => a.time === timeValue)) {
                timeInput.style.textShadow = "0 0 10px red";
                setTimeout(() => timeInput.style.textShadow = "none", 500);
                return;
            }

            const newAlarm = {
                id: Date.now(),
                time: timeValue,
                active: true,
                goal: goalValue, // Save the goal
                label: goalValue || 'ALARM'
            };

            alarms.push(newAlarm);
            // Sort alarms by time
```

```
            alarms.sort((a, b) => a.time.localeCompare(b.time));

            saveAlarms();
            timeInput.value = ''; // Reset input
            goalInput.value = ''; // Reset goal input
        }

        function toggleAlarm(id) {
            const alarm = alarms.find(a => a.id === id);
            if (alarm) {
                alarm.active = !alarm.active;
                saveAlarms();
            }
        }

        function deleteAlarm(id) {
            alarms = alarms.filter(a => a.id !== id);
            saveAlarms();
        }

        function renderAlarms() {
            const list = document.getElementById('alarmsList');

            list.innerHTML = '';

            if (alarms.length === 0) {
                list.innerHTML = `
                    <div id="emptyState" class="absolute inset-0 flex flex-col items-center justify-center opacity-30 pointer-events-none">
                        <i class="far fa-clock fa-3x mb-4"></i>
                        <p class="tracking-widest">NO ALARMS SET</p>
                    </div>
                `;
                return;
            }

            alarms.forEach(alarm => {
                const div = document.createElement('div');
                div.className = \`flex flex-col p-4 mb-3 border \${alarm.active ? 'border-[#39ff14] bg-[#39ff14]/10' : 'border-gray-800 text-gray-600'} rounded transition-all duration-300 hover:neon-box-shadow group\`;

                div.innerHTML = \`
                    <div class="flex justify-between items-center">
                        <div class="flex flex-col">
                            <span class="text-3xl font-bold tracking-wider \${alarm.active ? 'neon-text' : ''}">\${alarm.time}</span>
                            <span class="text-[10px] tracking-[0.2em] uppercase \${alarm.active ? 'opacity-80' : 'opacity-40'}">
                                \${alarm.active ? 'ACTIVE' : 'INACTIVE'}
                                \${alarm.goal ? \` | GOAL: \${alarm.goal.substring(0, 20)}...\` : ''}
                            </span>
                        </div>
                        <div class="flex items-center gap-4">
                            <button onclick="toggleAlarm(\${alarm.id})" class="text-xl focus:outline-none transition-colors \${alarm.active ? 'text-[#39ff14]' : 'text-gray-700 hover:text-gray-500'}">
                                <i class="fas \${alarm.active ? 'fa-toggle-on' : 'fa-toggle-off'} fa-lg"></i>
                            </button>
                            <button onclick="deleteAlarm(\${alarm.id})" class="text-red-500 opacity-0 group-hover:opacity-100 transition-opacity focus:outline-none hover:text-red-400 ml-2">
                                <i class="fas fa-trash"></i>
                            </button>
                        </div>
                    </div>
                \`;
                list.appendChild(div);
            });
        }

        // --- Audio & Visualization Logic ---

        function initAudio() {
            if (!audioContext) {
                audioContext = new (window.AudioContext || window.webkitAudioContext)();
```

```
            }
            if (audioContext.state === 'suspended') {
                audioContext.resume();
            }

            if (!analyser) {
                analyser = audioContext.createAnalyser();
                analyser.fftSize = 2048; // Resolution of visualizer
            }
        }

        // Generate one cycle of sound based on selected tone
        function playSingleToneCycle() {
            initAudio();
            const now = audioContext.currentTime;

            oscillator = audioContext.createOscillator();
            gainNode = audioContext.createGain();

            // Connect: Oscillator -> Gain -> Analyser -> Destination
            oscillator.connect(gainNode);
            gainNode.connect(analyser);
            analyser.connect(audioContext.destination);

            if (currentTone === 'pulse') {
                // Digital Beep
                oscillator.type = 'square';
                oscillator.frequency.setValueAtTime(880, now);
                oscillator.frequency.setValueAtTime(1760, now + 0.1);

                gainNode.gain.setValueAtTime(0.5, now);
                gainNode.gain.setValueAtTime(0, now + 0.15);
                gainNode.gain.setValueAtTime(0.5, now + 0.25);
                gainNode.gain.setValueAtTime(0, now + 0.4);

                oscillator.start(now);
                oscillator.stop(now + 0.5);

            } else if (currentTone === 'raid') {
                // Siren / Raid alert
                oscillator.type = 'sawtooth';
                oscillator.frequency.setValueAtTime(400, now);
                oscillator.frequency.linearRampToValueAtTime(1000, now + 0.4);
                oscillator.frequency.linearRampToValueAtTime(400, now + 0.8);

                gainNode.gain.setValueAtTime(0, now);
                gainNode.gain.linearRampToValueAtTime(0.3, now + 0.1);
                gainNode.gain.linearRampToValueAtTime(0.3, now + 0.7);
                gainNode.gain.linearRampToValueAtTime(0, now + 0.8);

                oscillator.start(now);
                oscillator.stop(now + 0.8);

            } else if (currentTone === 'warp') {
                // Sci-fi Warp Drop
                oscillator.type = 'sine';
                oscillator.frequency.setValueAtTime(2000, now);
                oscillator.frequency.exponentialRampToValueAtTime(100, now + 0.6);

                gainNode.gain.setValueAtTime(0, now);
                gainNode.gain.linearRampToValueAtTime(0.5, now + 0.05);
                gainNode.gain.exponentialRampToValueAtTime(0.01, now + 0.6);

                oscillator.start(now);
                oscillator.stop(now + 0.6);

            } else if (currentTone === 'chime') {
                // Simple high-frequency chime
                oscillator.type = 'sine';
                oscillator.frequency.setValueAtTime(1000, now);
                gainNode.gain.setValueAtTime(0, now);
                gainNode.gain.linearRampToValueAtTime(0.4, now + 0.01);
                gainNode.gain.linearRampToValueAtTime(0, now + 0.2);
```

```
                oscillator.start(now);
                oscillator.stop(now + 0.3);

            } else if (currentTone === 'synth') {
                // Low, resonant triangle pulse
                oscillator.type = 'triangle';
                oscillator.frequency.setValueAtTime(150, now);
                gainNode.gain.setValueAtTime(0, now);
                gainNode.gain.linearRampToValueAtTime(0.6, now + 0.05);
                gainNode.gain.linearRampToValueAtTime(0.1, now + 0.3);
                gainNode.gain.linearRampToValueAtTime(0, now + 0.5);

                oscillator.start(now);
                oscillator.stop(now + 0.6);
            }
        }

        function playAlarmSound() {
            playSingleToneCycle();

            // Loop timing depends on tone length
            let duration = 800; // Default pulse
            if (currentTone === 'pulse') duration = 800;
            if (currentTone === 'raid') duration = 1000;
            if (currentTone === 'warp') duration = 700;
            if (currentTone === 'chime') duration = 500; // Chime is faster
            if (currentTone === 'synth') duration = 700; // Synth is medium

            intervalId = setTimeout(playAlarmSound, duration);
        }

        // --- Visualization Canvas ---
        function startVisualizer() {
            const canvas = document.getElementById('audioVisualizer');
            const ctx = canvas.getContext('2d');

            // Resize canvas to match display size
            canvas.width = canvas.offsetWidth;
            canvas.height = canvas.offsetHeight;

            const bufferLength = analyser.frequencyBinCount;
            const dataArray = new Uint8Array(bufferLength);

            function draw() {
                if(!isRinging) return; // Stop drawing if alarm stops

                animationFrameId = requestAnimationFrame(draw);

                analyser.getByteTimeDomainData(dataArray);

                ctx.fillStyle = 'rgba(0, 0, 0, 0.2)'; // Fade out effect
                ctx.fillRect(0, 0, canvas.width, canvas.height);

                ctx.lineWidth = 2;
                ctx.strokeStyle = '#ff3939'; // Red line for alarm
                ctx.beginPath();

                const sliceWidth = canvas.width * 1.0 / bufferLength;
                let x = 0;

                for(let i = 0; i < bufferLength; i++) {
                    const v = dataArray[i] / 128.0;
                    const y = v * canvas.height / 2;

                    if(i === 0) {
                        ctx.moveTo(x, y);
                    } else {
                        ctx.lineTo(x, y);
                    }

                    x += sliceWidth;
                }
```

```
                ctx.lineTo(canvas.width, canvas.height/2);
                ctx.stroke();
            }

            draw();
        }

        function checkAlarms(h, m, s) {
            if (s !== '00') return;

            const currentTimeStr = `${h}:${m}`;
            const matchingAlarm = alarms.find(a => a.time === currentTimeStr && a.active);

            if (matchingAlarm && !isRinging) {
                // Deactivate the alarm immediately so it doesn't trigger again next minute
                matchingAlarm.active = false;
                saveAlarms();
                triggerAlarm(matchingAlarm);
            }
        }

        function triggerAlarm(alarm) {
            isRinging = true;
            activeAlarm = alarm; // Set the active alarm
            initAudio();
            document.getElementById('ringingTime').innerText = alarm.time;

            // Show Modal
            const modal = document.getElementById('alarmModal');
            const ringingState = document.getElementById('ringingState');
            const focusState = document.getElementById('focusState');

            modal.classList.remove('hidden');
            ringingState.classList.remove('hidden');
            focusState.classList.add('hidden'); // Ensure focus state is hidden initially

            // Start Sound loop
            playAlarmSound();

            // Start Visualizer
            startVisualizer();

            window.focus();
        }

        function stopAlarm() {
            isRinging = false;

            // Stop Sound Loop
            clearTimeout(intervalId);
            cancelAnimationFrame(animationFrameId);

            if (oscillator) {
                try {
                    oscillator.stop();
                } catch(e) {}
            }

            const ringingState = document.getElementById('ringingState');
            const focusState = document.getElementById('focusState');
            const modal = document.getElementById('alarmModal');

            // Check if there is a goal to process
            if (activeAlarm && activeAlarm.goal) {
                // Transition to Focus State
                ringingState.classList.add('hidden');
                focusState.classList.remove('hidden');

                document.getElementById('alarmGoalDisplay').innerHTML = \`Goal: <span class="text-white">\${
activeAlarm.goal}</span>\`;
                document.getElementById('focusResult').classList.add('hidden');
                document.getElementById('focusLoading').classList.add('hidden');
```

```
                document.getElementById('generateFocusBtn').classList.remove('hidden');

            } else {
                // If no goal, just hide the modal
                modal.classList.add('hidden');
                activeAlarm = null;
            }
        }

        // --- Gemini API Logic ---
        async function fetchGeminiResponse(userQuery, systemPrompt) {
            try {
                const payload = {
                    contents: [{ parts: [{ text: userQuery }] }],
                    systemInstruction: {
                        parts: [{ text: systemPrompt }]
                    },
                };

                let response = null;
                let result = null;
                const maxRetries = 3;
                let delay = 1000; // 1 second delay

                for (let i = 0; i < maxRetries; i++) {
                    response = await fetch(apiUrl, {
                        method: 'POST',
                        headers: { 'Content-Type': 'application/json' },
                        body: JSON.stringify(payload)
                    });

                    if (response.ok) {
                        result = await response.json();
                        break;
                    } else if (i < maxRetries - 1) {
                        // Retry with exponential backoff
                        await new Promise(resolve => setTimeout(resolve, delay));
                        delay *= 2;
                    } else {
                        throw new Error(\`API request failed after \${maxRetries} attempts with status \${
    response.status}\`);
                    }
                }

                const text = result?.candidates?.[0]?.content?.parts?.[0]?.text;
                if (!text) throw new Error("API response was empty or malformed.");

                return text;

            } catch (error) {
                console.error("Gemini API Error:", error);
                return "ERROR: Focus generation failed. Connection to the grid unstable. Please try again.";
            }
        }

        async function initiateFocusGeneration() {
            if (!activeAlarm || !activeAlarm.goal) return;

            const goal = activeAlarm.goal;
            const loading = document.getElementById('focusLoading');
            const resultDiv = document.getElementById('focusResult');
            const button = document.getElementById('generateFocusBtn');

            button.classList.add('hidden');
            resultDiv.classList.add('hidden');
            loading.classList.remove('hidden');

            // Updated system prompt to ensure English output
            const systemPrompt = \`You are an AI assistant in a cyberpunk/neon-noir world, designed to help
    humans achieve their daily goals immediately upon waking. Generate a short, motivational quote (mantra) and
    three distinct, hyper-specific, actionable steps ("PROTOCOLS") to start the user's day, based on their goal.
    Format the response strictly using markdown for readability, with bold text for the mantra and a numbered
    list for the protocols. Keep the tone edgy, determined, and digital. The entire output must be in clear,
```

```
    direct English.\`;

            const userQuery = \`My primary goal for the day is: "\${goal}"\`;

            const generatedText = await fetchGeminiResponse(userQuery, systemPrompt);

            loading.classList.add('hidden');
            // Basic Markdown to HTML conversion (for bolding and line breaks)
            resultDiv.innerHTML = generatedText.replace(/\*\*(.*?)\*\*/g, '<span class="text-lg font-bold text-white neon-text">$1</span>').replace(/\n/g, '<br>');
            resultDiv.classList.remove('hidden');

            // Since the user is done, we can allow them to close the modal
            setTimeout(() => {
                button.innerHTML = '<i class="fas fa-check-circle mr-2"></i> FOCUS GENERATED';
                button.classList.remove('hidden');
                button.onclick = () => document.getElementById('alarmModal').classList.add('hidden');
            }, 500);
        }

        // Initial Render
        renderAlarms();

        // Audio Context Unlock for Browsers
        document.body.addEventListener('click', () => {
            initAudio();
        }, { once: true });

        // Add keydown event for 'Enter' on input
        document.getElementById('alarmTimeInput').addEventListener('keydown', (e) => {
            if (e.key === 'Enter') addAlarm();
        });
        document.getElementById('alarmGoalInput').addEventListener('keydown', (e) => {
            if (e.key === 'Enter') addAlarm();
        });

    </script>
</body>
</html>
```

Listing 1: Complete Source Code for Neon Alarm