EMG Core Analysis System - Technical Documentation

Complete breakdown of every component in index.html.

Table of Contents

HTML Structure

Lines 1-45: Head Section

Purpose: Document metadata, styling, and icon library setup

Key Elements:

<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>EMG Core Analysis System (101+ Persona)</title>

- Standard UTF-8 encoding
- Mobile-responsive viewport
- Descriptive title for browser tabs

External Dependencies:

<script src="https://cdn.tailwindcss.com"></script>
<link
href="[https://fonts.googleapis.com/css2?family=Inter:wght@400;500;600;700;800&display=

swap](https://fonts.googleapis.com/css2?family=Inter:wght@400;500;600;700;800&display=swap)">

- **Tailwind CSS**: Utility-first CSS framework (no custom CSS compilation needed)
- **Inter Font**: Modern, readable sans-serif typeface

Lucide Icons Integration (Lines 38-42):

```
<script type="module">
    import { createIcons, Search, Settings, Download, Zap, Users, X, CheckCheck }
    from '[https://unpkg.com/lucide@latest](https://unpkg.com/lucide@latest)';
    createIcons({ icons: { Search, Settings, Download, Zap, Users, X, CheckCheck } });
</script>
```

- ES6 module import from CDN
- Only loads required icons (performance optimization)
- createIcons() replaces <i data-lucide="..."> tags with SVG icons

Lines 13-37: Custom CSS

Dark Theme Base (Lines 14-17):

```
body, html {
    font-family: 'Inter', sans-serif;
    background-color: #0f172a; /* Slate 900 */
    color: #e2e8f0; /* Slate 200 */
}
```

- Forces dark mode regardless of system preference
- High-contrast text for readability

Custom Scrollbar (Lines 19-24):

```
::-webkit-scrollbar { width: 8px; }
::-webkit-scrollbar-track { background: #1e293b; }
::-webkit-scrollbar-thumb {
    background: #475569;
    border-radius: 4px;
}
```

- Webkit-only styling (Chrome, Edge, Safari)
- Matches dark theme aesthetic

- Slim 8px width for minimal intrusion

Button Styling (Lines 26-29):

```
.modern-btn {
    border-radius: 0.75rem;
    transition: all 0.2s;
    box-shadow: 0 4px 6px -1px rgba(0, 0, 0, 0.1), 0 2px 4px -2px rgba(0, 0, 0, 0.1);
}
```

- Consistent rounded corners (12px)
- Smooth hover transitions
- Subtle shadow for depth

Skeleton Loading Animation (Lines 34-40):

```
.skeleton-loading {
    background-color: #334155;
    animation: pulse 1.5s infinite;
}
@keyframes pulse {
    0%, 100% { opacity: 1; }
    50% { opacity: 0.5; }
}
```

- Visual feedback while AI generates response
- Gentle pulsing effect (not jarring)

Lines 46-104: Main UI Structure

Application Container (Line 48):

```
<div id="appContainer" class="max-w-6xl mx-auto space-y-8">
```

- Max width 1152px (readable on large screens)
- Centered with auto margins
- Vertical spacing between sections

Header (Lines 49-53):

```
<h1 class="text-4xl font-extrabold text-emerald-400 flex items-center justify-center">
    <i data-lucide="zap" class="w-8 h-8 mr-2"></i>
    EMG Collective Intelligence Engine
</h1>
```

- Zap icon symbolizes speed/power
- Emerald accent color (brand identity)

Configuration Panel (Lines 56-97):

- **Topic Input** (Lines 58-61): Multi-line textarea for complex queries
- **API Key Input** (Lines 66-69): Password field with red border when invalid
- **Perspective Selection Button** (Lines 73-78): Shows count of selected personas
- **Throttle Slider** (Lines 81-88): Range input with live value display

Critical Validation Logic:

```
<input type="password" id="apiKeyInput"
    class="...border border-red-500..."
    placeholder="Required for AI calls">
```

- Red border alerts user to missing requirement
- Removed via JavaScript when valid key entered (min 30 chars)

Lines 105-155: Results Section

Initially Hidden (Line 105):

```
<div id="resultsSection" class="space-y-6 hidden">
```

- hidden class prevents layout shift before analysis starts
- Revealed via JavaScript when user clicks "Start Analysis"

Progress Bar (Lines 107-117):

```
<div class="w-full bg-slate-700 rounded-full h-2.5">
  <div id="progressBar"
      class="bg-emerald-600 h-2.5 rounded-full transition-all duration-300 ease-out"
      style="width: 0%"></div>
</div>
```

- Width updated via JavaScript: progressBar.style.width = '${percent}%'
- CSS transition creates smooth animation
- Color changes: slate background → emerald fill

Synthesis Panel (Lines 120-142):

```
<div id="synthesisPanel" class="perspective-card p-6 hidden">
```

```
    <h2>...Collective Synthesis</h2>
    <p id="synthesisContent" class="...skeleton-loading">...</p>

    <div id="sourcePanel" class="...hidden">
        <ul id="sourceList"></ul>
    </div>

    <button id="downloadJsonBtn">Download JSON</button>
</div>
```

- Starts hidden, shown after synthesis completes
- synthesisContent displays final mega-analysis
- sourcePanel populated with grounded web sources
- Download button triggers JSON export

Perspective Grid (Lines 145-148):

```
<div id="perspectivesGrid" class="grid grid-cols-1 md:grid-cols-2 gap-6">
    <!-- Dynamically injected cards -->
</div>
```

- Responsive: 1 column mobile, 2 columns desktop
- Cards added via initializePersonaCards() function

Lines 151-165: Message Modal

Error/Warning Dialog (Lines 151-160):

```
<div id="messageModal" class="fixed inset-0 modal-backdrop hidden items-center
justify-center">
    <div class="bg-slate-700 rounded-xl p-6 max-w-sm">
        <h3 id="messageTitle" class="...text-red-400">Error</h3>
        <p id="messageContent"></p>
        <button id="closeMessageModal">Close</button>
    </div>
</div>
```

- Full-screen overlay (fixed inset-0)
- Semi-transparent backdrop (defined in CSS)
- Title color changes based on message type (red=error, yellow=warning)

Lines 168-200: Persona Selection Modal

Checkbox Interface (Lines 168-198):

```
<div id="perspectiveSelectionModal" class="...hidden">
  <div class="...h-[80vh] flex flex-col">
    <!-- Header with close button -->
    <div class="flex justify-between">
      <h3>Select AI Perspectives (101+)</h3>
      <button id="modalCloseBtn"><i data-lucide="x"></i></button>
    </div>

    <!-- Select All checkbox -->
    <input type="checkbox" id="selectAllCheckbox">

    <!-- Scrollable checkbox container -->
    <div id="personaCheckboxesContainer"
         class="flex-grow overflow-y-auto grid grid-cols-1 sm:grid-cols-2">
      <!-- Dynamically rendered checkboxes -->
    </div>

    <button id="closeModalFooterBtn">Done Selecting</button>
  </div>
</div>
```

- 80% viewport height with scroll
- 2-column grid on larger screens
- "Select All" master toggle
- Checkboxes rendered via JavaScript

CSS Styling System

Tailwind Utility Classes

Color Palette:

- slate-900 (#0f172a): Background
- slate-800 (#1e293b): Cards
- slate-700 (#334155): Inputs, borders
- emerald-500/600: Primary actions, progress
- blue-500/600: Analysis button
- red-500: Errors, warnings

Responsive Design:

- sm: prefix: ≥640px
- md: prefix: ≥768px

- lg: prefix: ≥1024px

Example:

Mobile: 1 column

Desktop: 2 columns

JavaScript Architecture

Lines 203-208: Configuration Constants

```
const PERSPECTIVES_DATA = {
   "First Principles Physicist": {
      description: "Applies first-principles physics reasoning...",
      promptModifier: "Provide a deep, comprehensive analysis..."
   },
   // ...
};
```

Structure:

- Object keys = Persona names
- Values = { description, promptModifier }
- promptModifier = System instruction sent to Gemini API

Prompt Engineering Pattern:

"Provide a deep, comprehensive analysis of the topic from the perspective of a '[PERSONA]'. The response must be approximately 250 lines long. Do not use markdown headers, lists, or formatting like bolding or italics, just continuous, flowing text to maximize length."

Key Design Decision: 250-line mandate forces dense prose, avoiding AI's natural tendency toward bullet points.

Lines 209-234: Programmatic Persona Generation

```
...Object.fromEntries(Array.from({length: 85}, (_, i) => {
   const personaName = `Synthetic Persona ${i + 17}`;
   return [personaName, {
      description: `A unique, expert-level AI persona focusing on niche area ${i + 17}.`,
```

```
        promptModifier: `Provide a deep, comprehensive analysis of the topic from the
                perspective of a '${personaName}'. The response must be
                approximately 250 lines long...`
    }];
}))
```

How It Works:

1. Array.from({length: 85}) creates array of 85 undefined elements
2. Map function generates persona object for each: Synthetic Persona 17, 18, ..., 101
3. Object.fromEntries() converts array of [key, value] pairs to object
4. Spread operator (...) merges with manually-defined personas

Why This Approach?:

- Avoids 85 copy-paste blocks
- Easy to adjust persona count (change length: 85)
- Maintains consistent structure

Lines 237-248: Global State

```
const Dom = {}; // Populated in initDomElements()
let isAnalysisRunning = false;
let selectedPersonas = [];
const allPersonas = Object.keys(PERSPECTIVES_DATA);

const API_MODEL = "gemini-2.5-flash-preview-05-20";
const API_URL = (apiKey) =>
`https://generativelanguage.googleapis.com/v1beta/models/${API_MODEL}:generateContent?
key=${apiKey}`;

let personaDelayMs = parseInt(localStorage.getItem('personaDelayMs') || '2000', 10);
let searchGroundingEnabled = JSON.parse(localStorage.getItem('searchGroundingEnabled')
|| 'true');
```

State Variables:

Variable	Type	Purpose
Dom	Object	Holds all DOM element references
isAnalysisRunning	Boolean	Prevents concurrent

		analyses
selectedPersonas	Array	List of active persona names
allPersonas	Array	All 101+ persona names (from keys)
personaDelayMs	Number	Throttle delay between API calls

localStorage Integration:

localStorage.getItem('personaDelayMs') || '2000'

- Retrieves saved setting or defaults to 2000ms
- Settings persist across browser sessions

Lines 251-274: DOM Initialization

```
function initDomElements() {
  Dom.topicInput = document.getElementById('topicInput');
  Dom.apiKeyInput = document.getElementById('apiKeyInput');
  Dom.startAnalysisBtn = document.getElementById('startAnalysisBtn');
  // ... 30+ element references
}
```

Purpose: Centralized DOM access

- Called once on page load
- Avoids repeated document.getElementById() calls
- Single source of truth for element references

Pattern:

Dom.elementName = document.getElementById('elementId');

Later access: Dom.elementName.textContent = 'value'

Lines 276-328: Event Handling

API Key Validation (Lines 278-289):

```
Dom.apiKeyInput.addEventListener('input', () => {
    const key = Dom.apiKeyInput.value.trim();
    if (key.length > 30) {
        localStorage.setItem('geminiApiKey', key);
        Dom.apiKeyInput.classList.remove('border-red-500');
        Dom.startAnalysisBtn.disabled = false;
    } else {
        localStorage.removeItem('geminiApiKey');
        Dom.apiKeyInput.classList.add('border-red-500');
        Dom.startAnalysisBtn.disabled = true;
    }
});
```

Validation Logic:

- Gemini API keys are 39 characters, so > 30 is safe threshold
- Valid key: Remove red border, enable button, save to localStorage
- Invalid key: Red border, disable button, clear localStorage

Throttle Slider (Lines 294-300):

```
Dom.throttleRange.addEventListener('input', (e) => {
    const newDelay = parseInt(e.target.value, 10);
    personaDelayMs = newDelay;
    localStorage.setItem('personaDelayMs', newDelay);
    updateThrottleUI(newDelay);
});
```

- Real-time updates as user drags slider
- Saves immediately to localStorage
- Updates UI label and status text

Lines 331-403: Persona Selection Logic

Toggle Individual Persona (Lines 331-342):

```
function handlePersonaToggle(e) {
    const persona = e.target.value;
    if (e.target.checked) {
        if (!selectedPersonas.includes(persona)) {
            selectedPersonas.push(persona);
        }
    } else {
        selectedPersonas = selectedPersonas.filter(p => p !== persona);
```

```
    }
    localStorage.setItem('selectedPersonas', JSON.stringify(selectedPersonas));
    updateSelectedCountUI();
    updateSelectAllCheckbox();
}
```

Flow:

1. Get persona name from checkbox value
2. If checked: Add to array (if not already present)
3. If unchecked: Remove from array
4. Save to localStorage as JSON string
5. Update UI count display
6. Sync "Select All" checkbox state

Select All Toggle (Lines 344-352):

```
function handleSelectAllToggle(e) {
    if (e.target.checked) {
        selectedPersonas = [...allPersonas]; // Shallow copy
    } else {
        selectedPersonas = [];
    }
    localStorage.setItem('selectedPersonas', JSON.stringify(selectedPersonas));
    renderPersonaCheckboxes(); // Re-render to update all checkboxes
    updateSelectedCountUI();
}
```

Indeterminate State Logic (Lines 367-376):

```
function updateSelectAllCheckbox() {
    const allSelected = selectedPersonas.length === allPersonas.length;
    Dom.selectAllCheckbox.checked = allSelected;

    if (selectedPersonas.length > 0 && selectedPersonas.length < allPersonas.length) {
        Dom.selectAllCheckbox.indeterminate = true; // Dash icon
    } else {
        Dom.selectAllCheckbox.indeterminate = false; // Check or empty
    }
}
```

- All selected: Checkmark

- None selected: Empty
- Some selected: Dash (indeterminate)

Dynamic Rendering (Lines 378-402):

```
function renderPersonaCheckboxes() {
  Dom.personaCheckboxesContainer.innerHTML = '';

  allPersonas.forEach(persona => {
    const div = document.createElement('div');
    const isSelected = selectedPersonas.includes(persona);
    const checkboxId = `cb-${persona.replace(/[^a-zA-Z0-9]/g, '_')}`;

    div.innerHTML = `
      <input type="checkbox" id="${checkboxId}" value="${persona}"
          class="persona-checkbox ..." ${isSelected ? 'checked' : ''}>
      <label for="${checkboxId}">${persona}</label>
    `;
    Dom.personaCheckboxesContainer.appendChild(div);
  });

  // Add event listeners AFTER rendering
  document.querySelectorAll('.persona-checkbox').forEach(checkbox => {
    checkbox.addEventListener('change', handlePersonaToggle);
  });
}
```

Key Pattern: Render HTML string, then attach event listeners (avoids memory leaks from inline handlers)

Lines 406-457: UI Update Functions

Message Modal (Lines 406-422):

```
function showMessage(title, content, type = 'error') {
  Dom.messageTitle.textContent = title;
  Dom.messageContent.textContent = content;
  if (type === 'error') {
    Dom.messageTitle.classList.add('text-red-400');
    Dom.closeMessageModal.classList.add('bg-red-600');
  } else if (type === 'warning') {
    Dom.messageTitle.classList.add('text-yellow-400');
    Dom.closeMessageModal.classList.add('bg-yellow-600');
```

```
    }
    Dom.messageModal.classList.remove('hidden');
    Dom.messageModal.classList.add('flex');
}
```

Usage:

```
showMessage("API Key Missing", "Please enter a valid key", 'error');
showMessage("No Selection", "Select at least one persona", 'warning');
```

Throttle Status Text (Lines 424-437):

```
function updateThrottleStatus(ms) {
    let statusText;
    if (ms <= 1000) {
        statusText = 'Fast (High Risk, Low Latency)';
        Dom.throttleStatus.classList.add('text-red-400');
    } else if (ms <= 3000) {
        statusText = 'Standard (Medium Risk, Low Latency)';
        Dom.throttleStatus.classList.add('text-yellow-400');
    } else {
        statusText = 'Slow (Low Risk, High Latency)';
        Dom.throttleStatus.classList.add('text-green-400');
    }
    Dom.throttleStatus.textContent = statusText;
}
```

Risk Assessment:

- **≤1000ms**: May trigger rate limits
- **1000-3000ms**: Balanced
- **>3000ms**: Safe but slow

Initialize Persona Cards (Lines 446-457):

```
function initializePersonaCards(personas) {
    Dom.perspectivesGrid.innerHTML = '';
    personas.forEach((persona, index) => {
        const card = document.createElement('div');
        card.id = `perspective-card-${index}`;
        card.innerHTML = `
            <h3>${persona}</h3>
```

```
            <div id="perspective-content-${index}"
                 class="...skeleton-loading">
                Waiting for analysis...
            </div>
        `;
        Dom.perspectivesGrid.appendChild(card);
    });
}
```

Creates Skeleton Cards:

- One card per selected persona
- Unique ID for later content updates
- Pulsing loading animation

Update Card Content (Lines 459-466):

```
function updatePersonaCard(index, content) {
    const contentDiv = document.getElementById(`perspective-content-${index}`);
    if (contentDiv) {
        contentDiv.classList.remove('skeleton-loading');
        contentDiv.innerHTML = content.trim().replace(/\n/g, '<br>');
    }
}
```

- Removes loading animation
- Converts newlines to
 for proper display

Progress Bar Updates (Lines 468-473):

```
function updateProgressBar(completed, total) {
    const percent = Math.round((completed / total) * 100);
    Dom.progressText.textContent = `${completed}/${total} Perspectives Complete`;
    Dom.progressPercent.textContent = `${percent}%`;
    Dom.progressBar.style.width = `${percent}%`;
}
```

- Called after each persona completes
- Smooth CSS transition from previous width

API Integration

Lines 476-496: System Prompt Construction

```
function getSystemInstruction(personaName, topic) {
   const personaData = PERSPECTIVES_DATA[personaName];
   if (!personaData) {
      return `Analyze the topic: "${topic}". Provide a deep, comprehensive
            analysis. The response must be approximately 250 lines long...`;
   }
   return personaData.promptModifier.replace('the topic', `the topic: "${topic}"`);
}
```

Example Output:

Provide a deep, comprehensive analysis of the topic: "What is your first question if you perceive yourself as living?" from the perspective of a 'First Principles Physicist'. The response must be approximately 250 lines long. Do not use markdown headers, lists, or formatting...

Lines 498-509: Exponential Backoff

```
async function fetchWithExponentialBackoff(apiCall, maxRetries = 5, initialDelay = 1000) {
   for (let i = 0; i < maxRetries; i++) {
      try {
         return await apiCall();
      } catch (error) {
         if (i === maxRetries - 1) throw error;
         const delay = initialDelay * Math.pow(2, i) + Math.random() * 1000;
         await new Promise(resolve => setTimeout(resolve, delay));
      }
   }
}
```

Retry Schedule:

Attempt	Base Delay	Random Jitter	Total Delay
1	1000ms	0-1000ms	1-2s
2	2000ms	0-1000ms	2-3s
3	4000ms	0-1000ms	4-5s
4	8000ms	0-1000ms	8-9s
5	16000ms	0-1000ms	16-17s

Why Random Jitter?: Prevents "thundering herd" if multiple requests fail simultaneously

Lines 511-565: Generate Perspective

```
async function generatePerspective(personaName, topic, apiKey) {
    const systemPrompt = getSystemInstruction(personaName, topic);

    const payload = {
        contents: [{ parts: [{ text: `Topic: ${topic}` }] }],
        tools: [{ "google_search": {} }], // Enables grounding
        systemInstruction: {
            parts: [{ text: systemPrompt }]
        },
    };

    return await fetchWithExponentialBackoff(async () => {
        const response = await fetch(API_URL(apiKey), {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(payload)
        });

        if (!response.ok) {
            const errorBody = await response.json().catch(() => ({}));
            throw new Error(`API Error: ${response.status}...`);
        }

        const result = await response.json();
        const candidate = result.candidates?.[0];

        if (!candidate || !candidate.content?.parts?.[0]?.text) {
            throw new Error(`AI generation blocked for ${personaName}...`);
        }

        const text = candidate.content.parts[0].text;
        let sources = [];
        const groundingMetadata = candidate.groundingMetadata;
        if (groundingMetadata && groundingMetadata.groundingAttributions) {
            sources = groundingMetadata.groundingAttributions
                .map(attribution => ({
                    uri: attribution.web?.uri,
                    title: attribution.web?.title,
                }))
```

```
            .filter(source => source.uri && source.title);
        }

        return { perspective: text, sources };
    });
}
```

Request Structure:

```
{
  "contents": [{ "parts": [{ "text": "Topic: [user query]" }] }],
  "tools": [{ "google_search": {} }],
  "systemInstruction": {
    "parts": [{ "text": "[Persona-specific prompt]" }]
  }
}
```

Response Parsing:

1. Extract text from candidates[0].content.parts[0].text
2. Extract sources from groundingMetadata.groundingAttributions
3. Filter out sources without URI or title
4. Return object: { perspective, sources }

Lines 567-598: Generate Synthesis

```
async function generateSynthesis(fullAnalysis, apiKey) {
    const synthesisPrompt = `You are the EMG Collective Intelligence Synthesizer.
    Analyze the following collection of ${fullAnalysis.perspectives.length} diverse
    perspectives on: "${fullAnalysis.topic}".

    Identify core themes, consensus, conflicts, and emergent ideas.
    Do not summarize each one—synthesize a cohesive conclusion of ~250 lines.

    Perspectives for Synthesis:
    ${fullAnalysis.perspectives.map((p, i) =>
        `--- PERSPECTIVE ${i+1} (${p.persona}) ---\n${p.perspective}`
    ).join('\n\n')}`;

    const payload = {
        contents: [{ parts: [{ text: synthesisPrompt }] }],
        tools: [{ "google_search": {} }],
    };
```

```
    return await fetchWithExponentialBackoff(async () => {
        const response = await fetch(API_URL(apiKey), {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(payload)
        });
        // ... error handling ...
        return result.candidates?.[0]?.content?.parts?.[0]?.text ||
               "Synthesis failed: No content returned.";
    });
}
```

Mega-Prompt Structure:

You are the EMG Collective Intelligence Synthesizer...

--- PERSPECTIVE 1 (Persona Name) ---
[250 lines of analysis]

--- PERSPECTIVE 2 (Persona Name) ---
[250 lines of analysis]
...

The synthesis prompt is carefully crafted to be long enough to push the AI toward a deep, cross-cutting analysis, rather than a simple summary.

Analysis Execution Flow

1. **Validation**: Check for API Key, selected personas, and topic input.
2. **UI Setup**: Disable button, show resultsSection, initialize progressBar to 0, clear perspectivesGrid, and initialize skeleton cards.
3. Loop & Call: Iterate through selectedPersonas.
   a. Call generatePerspective: Use fetchWithExponentialBackoff to handle retries.
   b. Update UI: Call updatePersonaCard and updateProgressBar.
   c. Throttle: Wait for personaDelayMs after each successful call.
   d. Data Storage: Push results (persona, perspective, sources) into the fullAnalysisResults object.
4. **Synthesis**: After the loop, call generateSynthesis with the aggregated data.
5. **Final UI**: Update synthesisPanel with the final text and sources, enable the downloadJsonBtn.

6. **Cleanup**: Reset isAnalysisRunning and re-enable startAnalysisBtn.

Error Handling

- **API Key**: Input is disabled and bordered red until a valid key is provided (length > 30).
- **API Errors (4xx/5xx)**: Handled by fetchWithExponentialBackoff. Errors trigger a retry schedule (1s, 2s, 4s, 8s, 16s delay + jitter).
- **AI Safety Block/No Content**: If the AI returns no text (candidate check fails), a controlled error message is placed in the persona's card, and the loop continues, preventing a full crash.
- **User Input**: Validation for minimum topic length and persona selection prevents unnecessary API calls.
- **UI Error**: The showMessage modal handles user-facing errors (like missing key) without using alert().

Export Functionality

The downloadAnalysis() function handles the export.

1. **Data**: It uses the global fullAnalysisResults object, which contains the topic, timestamp, all individual perspectives (with their sources), and the final synthesis (with its sources).
2. **Format**: The data is stringified with a 2-space indentation (JSON.stringify(..., null, 2)).
3. **Download**: It creates a Blob, generates a temporary URL, and programmatically clicks a hidden anchor tag (<a>) to trigger the browser download. The file is named emg-analysis-YYYY-MM-DD.json.