

DueIntel\_MVP.html>

```
<html lang="en">
```

```
<head>
```

```
  <meta charset="UTF-8">
```

```
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
  <title>DueIntel | AI Due Diligence Tool</title>
```

```
  <!-- Load Tailwind CSS -->
```

```
  <script src="https://cdn.tailwindcss.com"></script>
```

```
  <!-- Load marked.js for Markdown rendering -->
```

```
  <script src="https://cdn.jsdelivr.net/npm/marked/marked.min.js"></script>
```

```
  <!-- Load Lucide Icons -->
```

```
  <script src="https://unpkg.com/lucide@latest"></script>
```

```
  <style>
```

```
    /* Custom styles for professional aesthetic and responsiveness */
```

```
    @import url('https://fonts.googleapis.com/css2?
```

```
family=Inter:wght@100..900&display=swap');
```

```
    body {
```

```
      font-family: 'Inter', sans-serif;
```

```
      background-color: #f7f7f9;
```

```
    }
```

```
    .report-card {
```

```
      box-shadow: 0 10px 30px rgba(0, 0, 0, 0.05);
```

```
      transition: all 0.3s ease;
```

```
    }
```

```
    .report-card:hover {
```

```
      box-shadow: 0 15px 40px rgba(0, 0, 0, 0.08);
```

```
    }
```

```
    /* Custom Markdown styling (rendered output) */
```

```
    .report-content h2 {
```

```
      font-size: 1.5rem;
```

```
      font-weight: 700;
```

```
      margin-top: 1.5rem;
```

```
      margin-bottom: 0.75rem;
```

```
      padding-bottom: 0.25rem;
```

```
      border-bottom: 2px solid #e0e0e0;
```

```
      color: #1f2937; /* Dark gray */
```

```
    }
```

```
    .report-content p, .report-content li {
```

```
      font-size: 1rem;
```

```
      line-height: 1.6;
```

```
      margin-bottom: 1rem;
```

```

        color: #374151; /* Medium gray */
    }
    .report-content ul {
        list-style-type: disc;
        padding-left: 1.5rem;
    }
    .report-content ul ul {
        list-style-type: circle;
    }

    /* Loading Spinner Animation */
    .spinner {
        border-top-color: transparent;
        border-left-color: #3b82f6; /* Blue 500 */
        animation: spin 1s ease-in-out infinite;
    }
    @keyframes spin {
        to { transform: rotate(360deg); }
    }
</style>
</head>
<body class="p-4 md:p-8">

    <div class="max-w-4xl mx-auto">
        <!-- Header -->
        <header class="mb-8 text-center">
            <h1 class="text-4xl font-extrabold text-gray-900 flex items-center justify-center">
                <i data-lucide="scan" class="w-8 h-8 mr-3 text-indigo-600"></i>
                DueIntel <span class="text-indigo-600 ml-1">.AI</span>
            </h1>
            <p class="mt-2 text-gray-500 text-lg">Rapid Financial Due Diligence for
Deal Screening</p>
        </header>

        <!-- Membership Status Card -->
        <div class="bg-gray-800 p-4 md:p-6 rounded-xl shadow-inner mb-6 text-white flex flex-col md:flex-row justify-between items-center">
            <div class="flex items-center space-x-3 mb-3 md:mb-0">
                <i data-lucide="pocket" class="w-6 h-6 text-green-400"></i>
                <div>
                    <span class="text-sm font-semibold text-gray-300">CURRENT
PLAN:</span>

```

```

        <p id="currentPlan" class="text-xl font-bold text-white"></p>
    </div>
</div>
<div class="flex items-center space-x-3">
    <i data-lucide="gauge" class="w-6 h-6 text-yellow-400"></i>
    <p class="text-lg font-medium">Analyses Remaining: <span
id="usageCount" class="font-bold text-yellow-300"></span></p>
    <!-- Simulation Buttons -->
    <button id="upgradeButton" class="ml-4 bg-green-500 hover:bg-
green-600 text-gray-900 font-semibold py-2 px-4 rounded-lg shadow-md
transition duration-200 ease-in-out text-sm flex items-center hidden"
onclick="simulateAnalystPlan()">
        <i data-lucide="trending-up" class="w-4 h-4 mr-1"></i>
        Simulate Analyst Tier
    </button>
</div>
</div>

<!-- Input Form Card -->
<div class="bg-white p-6 md:p-8 rounded-xl shadow-lg mb-8">
    <form id="dueDiligenceForm" class="flex flex-col md:flex-row gap-4">
        <input
            type="text"
            id="companyName"
            placeholder="Enter Company Name (e.g., 'Tesla' or 'Siemens')"
            required
            class="flex-1 p-3 border border-gray-300 rounded-lg focus:ring-
indigo-500 focus:border-indigo-500 transition duration-150 shadow-sm"
        >
        <button
            type="submit"
            id="analyzeButton"
            class="bg-indigo-600 hover:bg-indigo-700 text-white font-semibold
py-3 px-6 rounded-lg shadow-md transition duration-200 ease-in-out flex items-
center justify-center disabled:opacity-50"
        >
            <i data-lucide="bar-chart-3" class="w-5 h-5 mr-2"></i>
            Run Due Diligence
        </button>
    </form>
</div>

<!-- Report Output Area -->

```

```

<div id="reportContainer" class="hidden">
  <div class="flex justify-between items-center mb-6 border-b-2 border-
indigo-400 pb-2">
    <h2 class="text-3xl font-bold text-gray-800">
      <i data-lucide="file-text" class="w-6 h-6 inline mr-2 text-
indigo-600"></i>
      Due Diligence Report
    </h2>
    <!-- New Save Button (Visible for Analyst Tier) -->
    <button
      id="saveReportButton"
      class="hidden bg-green-600 hover:bg-green-700 text-white font-
semibold py-2 px-4 rounded-lg shadow-md transition duration-200 ease-in-out
text-sm flex items-center disabled:opacity-50"
    >
      <i data-lucide="save" class="w-4 h-4 mr-1"></i>
      Save Report
    </button>
  </div>

  <!-- Loading State -->
  <div id="loading" class="hidden flex flex-col items-center justify-center
bg-white p-12 rounded-xl shadow-lg report-card min-h-[300px]">
    <div class="spinner w-12 h-12 border-4 border-gray-200 rounded-full
mb-4"></div>
    <p class="text-gray-500 font-medium">Analyzing company
fundamentals and market data...</p>
    <p class="text-sm text-gray-400 mt-1">This may take a moment to
generate a grounded report.</p>
  </div>

  <!-- Report Content -->
  <div id="reportOutput" class="bg-white p-6 md:p-8 rounded-xl report-
card min-h-[300px] report-content">
    <!-- Content inserted here -->
  </div>

  <!-- Citations -->
  <div id="citations" class="mt-6 p-4 bg-gray-100 rounded-lg text-sm text-
gray-600 border border-gray-200 hidden">
    <p class="font-semibold mb-2">Sources of Information:</p>
    <ul id="sourceList" class="space-y-1"></ul>
  </div>

```

```
</div>
```

```
<!-- Error/Message Box -->
```

```
<div id="messageBox" class="hidden mt-8 p-4 bg-red-100 border border-red-400 text-red-700 rounded-lg" role="alert">
```

```
<!-- Error message inserted here -->
```

```
</div>
```

```
<!-- Footer Info -->
```

```
<div class="mt-8 pt-4 border-t text-xs text-gray-400 text-center">
```

```
<p id="authStatus"></p>
```

```
<p>App ID: <span id="appIdDisplay"></span></p>
```

```
</div>
```

```
</div>
```

```
<!-- Firebase Imports and Logic -->
```

```
<script type="module">
```

```
import { initializeApp } from "https://www.gstatic.com/firebasejs/11.6.1/firebase-app.js";
```

```
import { getAuth, signInAnonymously, signInWithCustomToken, onAuthStateChanged, setPersistence, browserSessionPersistence } from "https://www.gstatic.com/firebasejs/11.6.1/firebase-auth.js";
```

```
// New Firestore Imports for Saving Data
```

```
import { getFirestore, collection, addDoc, setLogLevel } from "https://www.gstatic.com/firebasejs/11.6.1/firebase-firestore.js";
```

```
// --- GLOBAL VARIABLES (Provided by Canvas Environment) ---
```

```
const appId = typeof __app_id !== 'undefined' ? __app_id : 'default-dueintel-app-id';
```

```
const firebaseConfig = typeof __firebase_config !== 'undefined' ? JSON.parse(__firebase_config) : null;
```

```
const initialAuthToken = typeof __initial_auth_token !== 'undefined' ? __initial_auth_token : null;
```

```
const apiKey = ""; // API key for Gemini
```

```
// Set up DOM elements
```

```
const form = document.getElementById('dueDiligenceForm');
```

```
const companyInput = document.getElementById('companyName');
```

```
const analyzeButton = document.getElementById('analyzeButton');
```

```
const reportContainer = document.getElementById('reportContainer');
```

```
const reportOutput = document.getElementById('reportOutput');
```

```

const loading = document.getElementById('loading');
const messageBox = document.getElementById('messageBox');
const citations = document.getElementById('citations');
const sourceList = document.getElementById('sourceList');
const appldDisplay = document.getElementById('appldDisplay');
const authStatus = document.getElementById('authStatus');
const usageCountEl = document.getElementById('usageCount');
const currentPlanEl = document.getElementById('currentPlan');
const upgradeButton = document.getElementById('upgradeButton');
const saveReportButton = document.getElementById('saveReportButton'); //

```

New Save Button

```

appldDisplay.textContent = appld;
setLogLevel('Debug');

```

```

let auth, db, userId = null; // db is new

```

```

let isAuthReady = false;

```

let lastReportContent = ""; // Global variable to hold the report content before saving

```

// --- USAGE AND PLAN MANAGEMENT (Simulated via localStorage) ---

```

```

const MAX_FREE_ANALYSES = 5;

```

```

const USAGE_KEY = 'dueintel_usage_count';

```

```

const PLAN_KEY = 'dueintel_plan'; // 'FREE' or 'ANALYST'

```

```

function getUsage() {

```

```

    // Check for the simulated Analyst key

```

```

    const plan = localStorage.getItem(PLAN_KEY) || 'FREE';

```

```

    const usedCount = parseInt(localStorage.getItem(USAGE_KEY)) || 0;

```

```

    let remaining = 0;

```

```

    let isLimited = false;

```

```

    if (plan === 'FREE') {

```

```

        remaining = MAX_FREE_ANALYSES - usedCount;

```

```

        isLimited = true;

```

```

    } else {

```

```

        // ANALYST or other paid plan

```

```

        remaining = '∞';

```

```

        isLimited = false;

```

```

    }

```

```

    return { plan, usedCount, remaining, isLimited };

```

```

}

// Function to simulate an upgrade
window.simulateAnalystPlan = function() {
  localStorage.setItem(PLAN_KEY, 'ANALYST');
  updateUsageDisplay();
  displayMessage('info', 'SUCCESS! You are now simulating the Analyst Tier.
You have unlimited reports and can now save reports!');
  saveReportButton.classList.add('hidden'); // Hide the button initially
}

function incrementUsage() {
  const { plan, usedCount, isLimited } = getUsage();
  if (isLimited && usedCount < MAX_FREE_ANALYSES) {
    localStorage.setItem(USAGE_KEY, usedCount + 1);
  }
}

function updateUsageDisplay() {
  const { plan, remaining, isLimited } = getUsage();

  // Display plan and usage
  currentPlanEl.textContent = plan === 'FREE' ? 'Explorer (Free Tier)' :
'Analyst (Pro Tier)';
  usageCountEl.textContent = remaining;

  // Handle button and input state based on limit
  const isLimitReached = isLimited && remaining <= 0;
  analyzeButton.disabled = isLimitReached;
  saveReportButton.classList.add('hidden'); // Hide save button on initial
display/reset

  if (isLimitReached) {
    displayMessage('info', `You have reached your Free Tier limit of $
{MAX_FREE_ANALYSES} analyses. Please upgrade to the Analyst tier for unlimited
access.`);
    upgradeButton.classList.remove('hidden');
    analyzeButton.classList.remove('bg-indigo-600', 'hover:bg-indigo-700');
    analyzeButton.classList.add('bg-gray-400', 'cursor-not-allowed');
  } else {
    hideMessage();
    upgradeButton.classList.add('hidden');
  }
}

```

```

        analyzeButton.classList.add('bg-indigo-600', 'hover:bg-indigo-700');
        analyzeButton.classList.remove('bg-gray-400', 'cursor-not-allowed');
    }
    // Show save button if plan is Analyst and a report exists
    if (plan !== 'FREE' && lastReportContent) {
        saveReportButton.classList.remove('hidden');
    }
    lucide.createIcons(); // Re-render icons after display update
}

```

// --- FIREBASE INITIALIZATION AND AUTHENTICATION ---

```

if (firebaseConfig) {
    const app = initializeApp(firebaseConfig);
    auth = getAuth(app);
    db = getFirestore(app); // Initialize Firestore

    // Set persistence to session to avoid constant re-auth
    setPersistence(auth, browserSessionPersistence);

    onAuthStateChanged(auth, async (user) => {
        if (user) {
            userId = user.uid;
            authStatus.textContent = `User authenticated: ${userId}`;
        } else {
            // Sign in anonymously if initial token is not available
            try {
                await signInAnonymously(auth);
                userId = auth.currentUser?.uid || 'anonymous';
                authStatus.textContent = `Signed in anonymously: ${userId}`;
            } catch (error) {
                console.error("Anonymous sign-in failed:", error);
                authStatus.textContent = "Authentication failed.";
            }
        }
        isAuthReady = true;
        updateUsageDisplay(); // Update UI after auth is ready
    });

    // Use custom token if provided
    if (initialAuthToken) {
        signInWithCustomToken(auth, initialAuthToken).catch((error) => {
            console.error("Custom token sign-in failed:", error);
        });
    }
}

```

```

    });
  }
} else {
  authStatus.textContent = "Firebase config missing. Running without auth/
db.";
  isAuthReady = true;
  updateUsageDisplay(); // Update UI even without Firebase
}

// --- FIRESTORE SAVING LOGIC (Analyst Tier Feature) ---

function getReportsCollectionRef() {
  // Private user-specific collection path for reports
  const collectionPath = `/artifacts/${apId}/users/${userId}/reports`;
  return collection(db, collectionPath);
}

async function saveReportToFirestore(companyName, content) {
  if (!db || !userId) {
    displayMessage('error', 'Database connection not ready. Please wait for
authentication.');
```

authentication.');

```

    return;
  }

  saveReportButton.disabled = true;
  saveReportButton.textContent = 'Saving...';

  try {
    const reportData = {
      company: companyName,
      contentMarkdown: content,
      createdAt: new Date().toISOString(),
      userId: userId,
      // Note: In a real app, you would also save the specific plan/tier here
    };

    const docRef = await addDoc(getReportsCollectionRef(), reportData);

    // Show success message without using alert()
    displayMessage('info', `Report for "${companyName}" saved
successfully! Document ID: ${docRef.id}`);

  } catch (error) {

```

```

        console.error("Error saving report to Firestore:", error);
        displayMessage('error', `Failed to save report: ${error.message}`);
    } finally {
        saveReportButton.disabled = false;
        saveReportButton.textContent = 'Save Report';
    }
}

// Event listener for the new save button
saveReportButton.addEventListener('click', () => {
    const companyName = companyInput.value.trim();
    if (lastReportContent && companyName) {
        saveReportToFirestore(companyName, lastReportContent);
    } else {
        displayMessage('error', 'No report content available to save.');
```

```

    });

    // --- GEMINI API LOGIC ---

```

```

    const model = 'gemini-2.5-flash-preview-09-2025';
    const apiUrl = `https://generativelanguage.googleapis.com/v1beta/models/${model}:generateContent?key=${apiKey}`;
    const MAX_RETRIES = 5;

```

```

    // Function to display messages/errors
    function displayMessage(type, text) {
        messageBox.className = 'mt-8 p-4 rounded-lg';
        if (type === 'error') {
            messageBox.classList.add('bg-red-100', 'border-red-400', 'text-red-700');
        } else {
            messageBox.classList.add('bg-blue-100', 'border-blue-400', 'text-blue-700');
        }
        messageBox.textContent = text;
        messageBox.classList.remove('hidden');
    }

```

```

    function hideMessage() {
        messageBox.classList.add('hidden');
    }

```

```

/**
 * Calls the Gemini API with exponential backoff.
 */
async function fetchWithBackoff(payload, attempt = 0) {
  try {
    const response = await fetch(apiUrl, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(payload)
    });

    if (response.status === 429 && attempt < MAX_RETRIES) {
      const delay = Math.pow(2, attempt) * 1000 + Math.random() * 1000;
      await new Promise(resolve => setTimeout(resolve, delay));
      return fetchWithBackoff(payload, attempt + 1);
    }

    if (!response.ok) {
      throw new Error(`API request failed with status ${response.status}`);
    }

    return response.json();
  } catch (error) {
    if (attempt < MAX_RETRIES) {
      const delay = Math.pow(2, attempt) * 1000 + Math.random() * 1000;
      await new Promise(resolve => setTimeout(resolve, delay));
      return fetchWithBackoff(payload, attempt + 1);
    }
    throw error;
  }
}

async function generateDueDiligenceReport(companyName) {
  const { plan } = getUsage();

  hideMessage();
  reportOutput.innerHTML = "";
  citations.classList.add('hidden');
  reportContainer.classList.remove('hidden');
  loading.classList.remove('hidden');
  analyzeButton.disabled = true;

```

```

lastReportContent = ''; // Clear previous content
saveReportButton.classList.add('hidden'); // Hide save button while
generating

let systemPrompt;
const userQuery = `Generate a structured financial due diligence report for
the company: ${companyName}.`;

if (plan === 'FREE') {
  // Free Tier Prompt: Limited report length, basic risk & overview only
  systemPrompt = `You are 'DuelIntel', an automated financial due diligence
assistant for the Explorer Free Tier. Your task is to generate a concise, standard
summary (150-200 words max) for the requested company based on public
information. The report MUST contain the following THREE sections exactly,
presented in brief Markdown format:
  1. '### Standard Overview' (What the company does)
  2. '### Key Risks & Basic Red Flags'
  3. '### Analyst Confidence Level'`;

  displayMessage('info', 'Generating Free Tier (Explorer) summary.
Upgrade to Analyst for full financial and ESG data.');
```

} else {
 // Analyst/Pro Tier Prompt: Full report (This is the report that can be
 saved)
 systemPrompt = `You are 'DuelIntel', a world-class, automated financial
 due diligence analyst specializing in rapid deal screening. Your task is to generate
 a professional, structured due diligence report for the requested company based
 on up-to-date, publicly available information. The report MUST contain the
 following SIX sections exactly, presented in detailed Markdown format. Use bullet
 points for detail where appropriate. The overall report should be professional and
 suitable for an initial investment screening, aiming for a length of approximately
 350-500 words of generated content:

1. '### Summary & Business Overview'
2. '### Key Financial Metrics' (Include recent Revenue, Net Income, P/E
 Ratio, and Market Cap.)
3. '### Recent Stock Performance & Market Data' (Include 1-Year Return
 percentage and major asset/stock-related news.)
4. '### Key Risks and Red Flags'
5. '### Governance & ESG Notes'
6. '### Analyst Confidence Level'`;
 }

```

const payload = {
  contents: [{ parts: [{ text: userQuery }] }],
  tools: [{ "google_search": {} }],
  systemInstruction: { parts: [{ text: systemPrompt }] },
};

try {
  const result = await fetchWithBackoff(payload);
  const candidate = result.candidates?.[0];

  if (candidate && candidate.content?.parts?.[0]?.text) {
    const text = candidate.content.parts[0].text;
    reportOutput.innerHTML = marked.parse(text); // Render Markdown
    lastReportContent = text; // Store raw Markdown for saving

    // Increment usage ONLY if it was a FREE tier analysis
    if (plan === 'FREE') {
      incrementUsage();
    } else if (plan === 'ANALYST') {
      // Show the save button only for Analyst tier
      saveReportButton.classList.remove('hidden');
    }

    // Extract and display grounding sources (citations)
    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);

      sourceList.innerHTML = ""; // Clear previous sources
      if (sources.length > 0) {
        sources.forEach((source, index) => {
          const li = document.createElement('li');
          li.innerHTML = `<a href="${source.uri}" target="_blank"
class="text-indigo-600 hover:text-indigo-800 underline transition
duration-150">(${index + 1}) ${source.title}</a>`;
          sourceList.appendChild(li);

```

```

        });
        citations.classList.remove('hidden');
    } else {
        citations.classList.add('hidden');
    }
} else {
    citations.classList.add('hidden');
}

} else {
    reportOutput.innerHTML = '<p class="text-red-500 font-
semibold">Error: Could not generate report. The model returned no text
content.</p>';
    lastReportContent = "";
}

} catch (error) {
    console.error("Due Diligence API error:", error);
    displayMessage('error', `Report generation failed: ${error.message}.
Please check the company name and try again.`);
    reportOutput.innerHTML = "";
} finally {
    loading.classList.add('hidden');
    analyzeButton.disabled = false;
    updateUsageDisplay(); // Final check for disabling button & showing save
button
}
}

// --- EVENT LISTENER ---

form.addEventListener('submit', function(e) {
    e.preventDefault();
    const companyName = companyInput.value.trim();
    const { remaining } = getUsage();

    if (!isAuthReady) {
        displayMessage('error', 'Authentication is still initializing. Please wait a
moment and try again.');
```

return;

```

    }

    if (getUsage().plan === 'FREE' && remaining <= 0) {
```

```
        displayMessage('error', `Analysis limit reached for the Free Tier. Please
upgrade.`);
        return;
    }

    if (companyName) {
        generateDueDiligenceReport(companyName);
    } else {
        displayMessage('error', 'Please enter a valid company name.');
```

```
    }
});

// Initialize icons and usage display on load
window.onload = () => {
    // We call this last to ensure all logic is loaded before calling createIcons
    updateUsageDisplay();
};
```

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