```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Creative Prompt Lab: Custom Modifiers</title>
    <script src="https://cdn.tailwindcss.com"></script>
    <style>
        @import
url('https://fonts.fonts.googleapis.com/css2?family=Inter:wght@100..90
0&display=swap');
        body {
            font-family: 'Inter', sans-serif;
            background-color: #0d1117;
            color: #c9d1d9;
        }
        .scrollable-category {
            max-height: 250px;
            overflow-y: auto;
            /* Simple scrollbar styling for dark mode */
            scrollbar-width: thin;
            scrollbar-color: #30363d #161b22;
        }
        .scrollable-category::-webkit-scrollbar {
            width: 8px;
        }
        .scrollable-category::-webkit-scrollbar-track {
            background: #161b22;
        }
        .scrollable-category::-webkit-scrollbar-thumb {
            background-color: #30363d;
            border-radius: 20px;
            border: 2px solid #161b22;
        }
        .pill {
            cursor: pointer;
            transition: all 0.2s;
            user-select: none;
        }
        .pill:hover {
            opacity: 0.8;
            transform: translateY(-1px);
        }
        .pill.selected {
            background-color: #238636; /* GitHub green */
            color: #ffffff;
            border-color: #238636;
```

```
}
/* Custom Status Box Styling */
#statusMessage {
    transition: opacity 0.3s ease-in-out;
    border-radius: 0.75rem; /* Rounded-xl */
    padding: 1rem;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.2);
    font-weight: 500;
}
.status-error {
    background-color: #991b1b; /* Red-700 */
    color: #fee2e2; /* Red-100 */
    border: 1px solid #dc2626; /* Red-600 */
}
.status-success {
    background-color: #166534; /* Green-700 */
    color: #dcfce7; /* Green-100 */
    border: 1px solid #22c55e; /* Green-500 */
}

/* Custom Toggle Switch for Expander */
.toggle-switch {
    position: relative;
    display: inline-block;
    width: 60px;
    height: 34px;
}

.toggle-switch input {
    opacity: 0;
    width: 0;
    height: 0;
}

.slider {
    position: absolute;
    cursor: pointer;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    background-color: #30363d;
    transition: .4s;
    border-radius: 34px;
}

.slider:before {
    position: absolute;
```

```
            content: "";
            height: 26px;
            width: 26px;
            left: 4px;
            bottom: 4px;
            background-color: white;
            transition: .4s;
            border-radius: 50%;
        }

        input:checked + .slider {
            background-color: #238636; /* Green */
        }

        input:checked + .slider:before {
            transform: translateX(26px);
        }
    </style>
</head>
<body class="p-4 md:p-8">

    <div id="app" class="max-w-7xl mx-auto">
        <header class="mb-8 border-b border-gray-700 pb-4">
            <h1 class="text-4xl font-extrabold text-white
text-center">🧪 Creative Prompt Lab: Custom Modifiers</h1>
            <p class="text-center mt-2 text-gray-400">Craft your
masterpiece with unlimited creative control and personalized
options.</p>
        </header>

        <main class="grid grid-cols-1 lg:grid-cols-3 gap-8">
            <div class="lg:col-span-2 space-y-6">

                <section class="bg-[#161b22] p-4 rounded-xl shadow-lg
border border-[#30363d] space-y-4">
                    <h2 class="text-xl font-semibold text-white">1.
Core Concept Prompt</h2>
                    <textarea id="basePrompt" rows="3" class="w-full
p-3 bg-[#0d1117] border border-[#30363d] rounded-lg
focus:ring-blue-500 focus:border-blue-500 text-white resize-none"
placeholder="A majestic griffin soaring over a neon-lit, futuristic
Tokyo cityscape..."></textarea>

                    <div class="mt-4 border-t border-gray-700 pt-4">
                        <div class="flex justify-between items-center
mb-2">
                            <h3 class="text-lg font-medium
text-gray-300">💡 Quick Prompt Ideas (Click to load)</h3>
```

```
                            <button id="refreshPromptsBtn"
onclick="refreshPrompts()" class="flex items-center space-x-1 p-2
text-sm rounded-lg bg-[#30363d] border border-[#40464d] text-gray-300
hover:bg-[#40464d] transition duration-200">
                                <svg
xmlns="http://www.w3.org/2000/svg" width="16" height="16" viewBox="0 0
24 24" fill="none" stroke="currentColor" stroke-width="2"
stroke-linecap="round" stroke-linejoin="round" class="w-4 h-4"><path
d="M21.5 2v6h-6"/><path d="M21.5 8l-2.414-2.414C17.5 3.737 14.837 2 12
2c-5.523 0-10 4.477-10 10s4.477 10 10 10 10-4.477 10-10"/></svg>
                                <span>Refresh Ideas</span>
                            </button>
                        </div>
                        <div id="promptSuggestions" class="space-y-2">
                            </div>
                    </div>

                    <div class="flex flex-wrap gap-3 items-center
border-t border-gray-700 pt-4">
                        <label class="text-sm font-medium">Aspect
Ratio:</label>
                        <div id="aspectRatioContainer" class="flex
flex-wrap gap-2">
                            </div>
                    </div>

                    <div class="mt-4 border-t border-gray-700 pt-4">
                        <h3 class="text-lg font-medium text-white
mb-2">Generate Prompt from Image</h3>
                        <input type="file" id="imageUpload"
accept="image/*" class="w-full text-sm text-gray-400 file:mr-4
file:py-2 file:px-4 file:rounded-full file:border-0 file:text-sm
file:font-semibold file:bg-[#30363d] file:text-white
hover:file:bg-[#40464d] cursor-pointer"/>
                        <button onclick="uploadAndGeneratePrompt()"
id="uploadPromptBtn" class="mt-3 w-full bg-green-600
hover:bg-green-700 text-white font-bold py-2 px-4 rounded-lg
transition duration-200 shadow-md">
                            Analyze Image & Generate Prompt
                        </button>
                    </div>
                </section>

                <section class="bg-[#161b22] p-4 rounded-xl shadow-lg
border border-[#30363d] space-y-6">
                    <h2 class="text-xl font-semibold text-white">2.
Creative Modifiers (8 Categories)</h2>
                    <p class="text-sm text-gray-400">Select options
```

```
below to build a hyper-detailed prompt. (Exclusive selection for
Style, Artist, Lighting). **Use the fields below to permanently add
your own options!**</p>

                    <div id="categoryContainer" class="grid
md:grid-cols-2 gap-6">
                        <div id="loadingCategories"
class="lg:col-span-2 flex justify-center items-center p-8
text-blue-400">
                            Loading modifier data...
                        </div>
                    </div>
                </section>

                <div class="bg-[#0d1117] p-4 rounded-xl border
border-[#30363d] flex justify-between items-center shadow-lg">
                    <div>
                        <h3 class="text-xl font-bold text-blue-400
flex items-center">
                            <svg xmlns="http://www.w3.org/2000/svg"
width="24" height="24" viewBox="0 0 24 24" fill="none"
stroke="currentColor" stroke-width="2" stroke-linecap="round"
stroke-linejoin="round" class="w-6 h-6 mr-2"><circle cx="12" cy="12"
r="10"/><path d="M8 12h8"/><path d="M12 8v8"/></svg>
                            Prompt Expander: Hyper-Detail Mode
                        </h3>
                        <p class="text-sm text-gray-500 mt-1">Activate
for complex, cinematic, and technically perfect AI-engineered
prompts.</p>
                    </div>
                    <label class="toggle-switch">
                        <input type="checkbox"
id="promptExpanderToggle">
                        <span class="slider"></span>
                    </label>
                </div>

                <button onclick="generateArt()" id="generateBtn"
class="w-full bg-blue-600 hover:bg-blue-700 text-white font-extrabold
py-4 px-4 rounded-xl text-xl shadow-2xl transition duration-300">
                    GENERATE AI ART
                </button>
            </div>

            <div class="lg:col-span-1 space-y-6">
                <section class="bg-[#161b22] p-4 rounded-xl shadow-lg
border border-[#30363d] h-full flex flex-col">
                    <h2 class="text-xl font-semibold text-white
```

```
mb-4">3. Generated Output</h2>
                    <div id="outputArea" class="flex-grow flex
items-center justify-center bg-[#0d1117] rounded-lg overflow-hidden
min-h-[300px] relative">
                        <div id="loadingIndicator" class="hidden
absolute inset-0 bg-gray-900 bg-opacity-70 flex flex-col items-center
justify-center p-4">
                            <svg class="animate-spin -ml-1 mr-3 h-8
w-8 text-blue-500" xmlns="http://www.w3.org/2000/svg" fill="none"
viewBox="0 0 24 24">
                                <circle class="opacity-25" cx="12"
cy="12" r="10" stroke="currentColor" stroke-width="4"></circle>
                                <path class="opacity-75"
fill="currentColor" d="M4 12a8 8 0 018-8V0C5.373 0 0 5.373 0 12h4zm2
5.291A7.962 7.962 0 014 12H0c0 3.042 1.135 5.824 3
7.938l3-2.647z"></path>
                            </svg>
                            <p class="mt-3 text-white
text-center">Loading...</p>
                            <p id="loadingTip" class="text-xs
text-gray-400 mt-1"></p>
                        </div>
                        <img id="generatedImage" class="hidden w-full
h-auto rounded-lg object-contain"
src="https://placehold.co/600x600/161b22/c9d1d9?text=Your+Art+Appears+
Here" alt="Generated AI Art" onerror="this.onerror=null;
this.src='https://placehold.co/600x600/161b22/c9d1d9?text=Error+Loadin
g+Image'" />
                        <p id="placeholderText" class="text-gray-500
p-4 text-center">Click 'GENERATE AI ART' to see your creation.</p>
                    </div>

                    <button id="downloadBtn"
onclick="handleDownload()" class="hidden w-full bg-orange-500
hover:bg-orange-600 text-white font-bold py-3 mt-4 rounded-lg text-lg
transition duration-200 shadow-md">
                        📥 DOWNLOAD IMAGE (Save to Device)
                    </button>

                    <div id="statusMessage" class="mt-4 p-3 hidden
text-sm rounded-lg"></div>
                </section>
            </div>
        </main>
    </div>

    <script type="module">
        // --- Firebase Imports ---
```

```
        import { initializeApp } from
"https://www.gstatic.com/firebasejs/11.6.1/firebase-app.js";
        import { getAuth, signInAnonymously, signInWithCustomToken,
setPersistence, browserLocalPersistence } from
"https://www.gstatic.com/firebasejs/11.6.1/firebase-auth.js";
        // NOTE: Use updateDoc for targeted updates instead of
overwriting the whole document with setDoc
        import { getFirestore, doc, getDoc, setDoc, onSnapshot,
setLogLevel, updateDoc } from
"https://www.gstatic.com/firebasejs/11.6.1/firebase-firestore.js";

        // --- API Setup (Necessary for Gemini API access) ---
        const apiKey = ""; // API Key is provided by the environment
        const GENERATE_ART_URL =
`https://generativelanguage.googleapis.com/v1beta/models/imagen-3.0-ge
nerate-002:predict?key=${apiKey}`;
        const GENERATE_PROMPT_URL =
`https://generativelanguage.googleapis.com/v1beta/models/gemini-2.5-fl
ash-preview-05-20:generateContent?key=${apiKey}`;
        const EXPANDER_URL =
`https://generativelanguage.googleapis.com/v1beta/models/gemini-2.5-fl
ash-preview-05-20:generateContent?key=${apiKey}`; // New URL for
prompt expansion

        // Global Firebase Variables (Provided by environment)
        const appId = typeof __app_id !== 'undefined' ? __app_id :
'default-app-id';
        const firebaseConfig = typeof __firebase_config !==
'undefined' ? JSON.parse(__firebase_config) : {};
        const initialAuthToken = typeof __initial_auth_token !==
'undefined' ? __initial_auth_token : null;

        let db;
        let auth;
        let userId = 'loading'; // Will be updated on successful
sign-in
        let CATEGORY_DATA = {}; // Dynamic data structure loaded from
Firestore
        let selectedOptions = {};
        let currentSamplePrompts = []; // Stores the currently
displayed random prompts

        // --- Data storage for image generation ---
        let lastGeneratedJpgBase64Data = null;

        // --- Full Library of Prompts for Randomization ---
        const FULL_PROMPT_LIBRARY = [
            "A lonely space cowboy riding a cosmic horse through a
```

```
nebula, cinematic lighting, vaporwave style.",
            "An ornate, brass-and-wood steampunk airship docking over
a misty Victorian London skyline, hyperdetailed oil painting.",
            "A highly detailed, bioluminescent deep-sea creature
swimming through underwater ruins, Rule of Thirds composition.",
            "A post-apocalyptic nature reclaiming a desolate, rusted
robot, matte painting, hyperdetailed cinematic.",
            "A futuristic ramen stall illuminated by neon signs in a
rainy Tokyo alley, film noir style, moody.",
            "An ancient Greek marble statue shattered by digital
glitch effects, with a feeling of chaotic beauty.",
            "A floating celestial library orbiting a purple gas giant,
detailed with God rays and ethereal glow.",
            "A tiny cottage nestled inside the roots of a giant,
magical tree, drawn by Hayao Miyazaki.",
            "A dynamic portrait of a cyberpunk samurai with chrome
armor, rendered in vibrant RGB light.",
            "A majestic griffin soaring over a neon-lit, futuristic
Tokyo cityscape, drawn by Syd Mead in the style of vaporwave.",
            "A tiny astronaut standing on a giant, moss-covered
ringworld, cold blue moonlight, cinematic dramatic lighting.",
            "A serene and calm portrait of an ancient robotic shaman
holding a staff, oil painting style, volumetric lighting.",
            "Post-apocalyptic nature reclaiming a desolate skyscraper,
matte painting, hyperdetailed cinematic, melancholy and lonely.",
            "An abstract geometric arrangement of vibrant, glowing
jellyfish swimming in a celestial void, prismatic distortion.",
            "A medieval knight facing a massive, spectral dragon in a
blizzard, high contrast B&W, extreme close-up.",
            "A tranquil, low-poly 3D scene of a desert oasis at
sunrise, with warm golden hour light, rule of thirds.",
            "A photorealistic close-up of a shattered hourglass filled
with stardust, deep focus, extreme detail.",
            "An Art Nouveau poster of a woman entwined with luminous
flora, inspired by Alphonse Mucha, golden ratio.",
            "A vibrant Pop Art illustration of a comic book hero
saving a cat from a falling satellite, dynamic angle.",
            "A desolate, wind-swept beach featuring the ruins of a
collapsed space elevator, rendered with film grain and a sepia tone."
        ];


        // --- Initial Seed Data (Used only once to populate the
database) ---
        const SEED_DATA = {
            artist: {
                title: "Artist Inspiration",
                prefix: "Art by",
```

```
                options: [
                    "Greg Rutkowski (ArtStation)", "Zdislav Beksinski
(Surrealism)", "Alphonse Mucha (Art Nouveau)",
                    "Hayao Miyazaki (Ghibli)", "Leonardo da Vinci
(Renaissance)", "Frank Frazetta (Fantasy)",
                    "Moebius (Sci-Fi Comics)", "Peter Mohrbacher
(Angelarium)", "Loish (Digital Portrait)",
                    "Beeple (Digital Abstract)", "Banksy (Street
Art)", "H.R. Giger (Biomechanical)",
                    "Claude Monet (Impressionism)", "Victo Ngai
(Illustrative)", "Escher (Impossible Geometry)",
                    "Van Gogh (Post-Impressionism)", "Syd Mead
(Futurism)", "Klimt (Symbolism)",
                    "NightCafe Stylized", "Midjourney V6 Aesthetic",
"Makoto Shinkai (Anime Landscapes)",
                    "Ivan Aivazovsky (Seascapes)", "Hokusai
(Woodblock)", "Caravaggio (Baroque Tenebrism)",
                    "James Gurney (Dinotopia)", "Simon Stålenlenhag
(Retro Sci-Fi)", "Ghibli Studio Aesthetic",
                    "Disney Concept Artist", "Pixar Aesthetic", "Chris
Foss (Spaceships)",
                    "John Berkey (Space Art)", "J.C. Leyendecker
(Illustration)", "Norman Rockwell (Americana)",
                    "Albert Bierstadt (Romantic Landscapes)", "William
Blake (Symbolism)", "Egon Schiele (Expressionism)",
                    "Jean-Michel Basquiat (Neo-Expressionism)",
"Takashi Murakami (Superflat)", "Yayoi Kusama (Polka Dots)",
                    "Shepard Fairey (Poster Art)", "Art Deco Master",
"Bauhaus Style", "Rococo Elegance",
                    "Post-Impressionist Master", "Abstract Geometric
Artist", "Photorealist Painter",
                    "Hyperrealist Oil Painter", "Concept Art Sketch
Artist", "Matte Painting Digital Master"
                ]
            },
            style: {
                title: "Art Style",
                prefix: "in the style of",
                options: [
                    "Steampunk", "Photorealistic", "Oil Painting",
"Abstract Expressionism", "Low-Poly 3D",
                    "Anime Style", "Ukiyo-e Woodblock", "Voxel Art",
"Hyperdetailed Cinematic",
                    "Synthwave", "Cyberpunk", "Pixel Art", "Charcoal
Sketch", "Watercolor",
                    "Matte Painting", "Conceptual Art", "Dystopian
Grunge", "Gothic Architecture",
                    "Pop Art", "Film Noir", "Gouache Painting",
```

```
"Pastel Drawing", "Pencil Sketch",
                    "Marker Illustration", "Mosaic Art", "Stained
Glass", "Holographic", "Glitchcore",
                    "Vaporwave", "Neo-Expressionist", "Maximalism",
"Minimalism", "Brutalist Architecture",
                    "Trompe-l'oeil", "Daguerreotype Photo", "Wet Plate
Collodion", "Cyanotype Print",
                    "Lithograph", "Etching", "Mecha Design",
"Biopunk", "Solarpunk", "Dieselpunk",
                    "Toy Art", "Graffiti Style", "Mandala Pattern",
"Zine Aesthetic", "Paper Craft",
                    "Claymation", "Pixelated Glitch", "Fluid
Dynamics", "Abstract Landscape", "Ink Splatter"
                ]
            },
            mood: {
                title: "Mood & Emotion",
                prefix: "with a feeling of",
                options: [
                    "Eerie and haunting", "Serene and calm", "Chaotic
and kinetic", "Melancholy and lonely",
                    "Joyful and vibrant", "Mysterious and ancient",
"Triumphant and bold", "Nostalgic and warm",
                    "Horror and dread", "Whimsical and playful",
"Aggressive and sharp", "Tranquil and meditative",
                    "Wondrous and epic", "Dreamlike and hazy",
"Intense and dramatic", "Cozy and intimate",
                    "Sublime and powerful", "Desolate and empty",
"Hopeful and bright", "Brooding and dark",
                    "Existential dread", "Crystalline clarity",
"Electric energy", "Quiet solitude",
                    "Savage intensity", "Ancient wisdom", "Weightless
tranquility", "Heavy atmosphere",
                    "Bitter defeat", "Tender affection", "Frenetic
panic", "Stoic patience", "Wistful longing",
                    "Chaotic excitement", "Deep satisfaction",
"Haunting beauty", "Playful mischief",
                    "Cold indifference", "Warm embrace", "Spiritual
awakening", "Sensory overload",
                    "Muted reflection", "Bold confrontation", "Subtle
threat", "Overwhelming scale"
                ]
            },
            theme: {
                title: "Theme & Subject Focus",
                prefix: "with themes of",
                options: [
                    "Cyberpunk Cityscape", "Deep Sea Exploration",
```

```
"Ancient Mythology", "Space Opera Adventure",
                    "Post-Apocalyptic Nature", "Medieval Fantasy
Battle", "Haunted Mansion Interior", "Lost Alien Civilization",
                    "Robots in Nature", "Floating Islands", "Time
Travel Paradox", "Alchemist's Workshop",
                    "Zen Garden", "Digital Glitchscape", "Cyborg
Portrait", "Luminous Flora",
                    "Gilded Age Elegance", "Victorian Sci-Fi", "Cosmic
Horror", "Future Fashion",
                    "Norse Mythology", "Japanese Folklore", "Desert
Nomads", "Deep Space Anomaly",
                    "Underwater Ruins", "Bio-luminescent Forest",
"Time-Worn Clockwork", "Magitech",
                    "Floating Marketplace", "Sentinel Robots", "Lost
Library", "Post-Human Earth",
                    "Giant Kaiju", "Secret Garden", "Neon Alleyways",
"Volcanic Eruption",
                    "Arctic Research Base", "Subterranean City",
"Aerial View of a Storm", "Inside a Microchip",
                    "Symbiotic Creatures", "Quantum Entanglement",
"Mythical Creature Taxonomy", "Ritualistic Objects",
                    "Evolving Landscapes", "The Last Human", "Alien
Flora", "Haunted Carnival"
                ]
            },
            lighting: {
                title: "Lighting & Color",
                prefix: "illuminated by",
                options: [
                    "Volumetric lighting", "Cinematic dramatic
lighting", "God rays", "Rim lighting",
                    "Neon glow", "Twilight hour", "Studio softbox",
"Ethereal glow",
                    "Overcast day", "Harsh sunlight", "Bioluminescent
light", "Misty and diffused",
                    "Backlit silhouette", "Warm golden hour", "Cold
blue moonlight", "Candlelight flicker",
                    "Vibrant RGB light", "Ultraviolet light",
"Chromatic aberration", "Dynamic range HDR",
                    "Spotlight", "Soft Ambient", "Harsh Contrast",
"Low Key", "High Key", "Blue Hour",
                    "Subsurface Scattering", "Fluorescent Tubes",
"Muted Palette", "Monochromatic",
                    "Triadic Colors", "Analog Colors", "Complimentary
Colors", "Split-Complementary",
                    "Warm Tones", "Cool Tones", "Infrared Vision",
"X-ray Effect", "Light Pillars",
                    "Ray Tracing", "Opaque Shadows", "Refraction",
```

```
"Caustics", "Volumetric Fog",
                    "Chromatic Spectrum", "High Saturation",
"Desaturated", "Muted Colors", "Edge Glow"
                ]
            },
            background: {
                title: "Background & Environment",
                prefix: "set against a background of",
                options: [
                    "Bokeh effect", "Vast Nebula cloud", "Simple
gradient", "Cracked Earth wasteland",
                    "Overgrown Ruins", "Distant Mountain Range", "Void
of Space", "Dense Jungle canopy",
                    "Flooded City streets", "Mirror Lake reflection",
"Infinite White space", "Digital Circuit board",
                    "Steaming Volcano", "Sand Dunes at sunset",
"Glacial landscape", "Empty Desert road",
                    "Woven tapestry", "Heavy rain and fog", "Crystal
formations", "Abstract geometric pattern",
                    "Distant Metropolis", "Moonlit Ocean", "Infinite
Desert", "Dense Fog",
                    "Abstract Particles", "Corroded Metal", "Winding
River", "Crystal Cave",
                    "Lush Valley", "Tectonic Plates", "Ice Cavern",
"Submerged Temple",
                    "Bamboo Forest", "Snow Covered Tundra", "Shifting
Sand Dunes", "Cloud Sea",
                    "Aurora Borealis", "Gilded Frame", "Wrought Iron
Fence", "Moss Covered Stone",
                    "Electric Storm", "Zero Gravity", "Floating
Debris", "Geometric Patterned Tiles",
                    "Wall of Monitors", "Cracked Glass", "Rainbow
Mist", "Cosmic Ripples"
                ]
            },
            filters: {
                title: "Filters & Post-Processing",
                prefix: "with effects like",
                options: [
                    "Film grain", "Sepia tone", "Glitch effect",
"Subtle lens flare",
                    "Vibrant colors", "Sketch lines", "Halftone
print", "High contrast B&W",
                    "Anamorphic lens blur", "Double exposure",
"Chromatic noise", "Depth of Field (DOF)",
                    "Polaroid photo", "Cross-hatch shading", "Digital
oil paint filter", "Aged parchment",
                    "HDR processing", "Soft focus dreamy", "Monochrome
```

```
cyanotype", "Prismatic distortion",
                    "Tilt-shift", "Fisheye Lens", "Wide Angle",
"Telephoto Compression",
                    "VHS Filter", "Scanline Effect", "Old Film Look",
"Grainy Texture",
                    "Cross Processed", "Bleach Bypass", "Solarized
Effect", "Gaussian Blur",
                    "Motion Blur", "Radial Blur", "Pixel Sorting",
"Data Loss Glitch",
                    "Distorted Geometry", "High Saturation",
"Desaturated", "Muted Colors",
                    "Shallow DOF", "Bokeh", "Edge Glow", "Chromatic
Aberration",
                    "Vignette", "Noise Reduction", "Super Resolution",
"Anti-aliasing"
                ]
            },
            composition: {
                title: "Composition & Angle",
                prefix: "composed with",
                options: [
                    "Rule of Thirds", "Golden Ratio spiral", "Dutch
Angle tilt", "Perfectly Symmetrical",
                    "Extreme Close-Up", "Ultra-Wide Shot", "Vertical
Pan perspective", "Tilt-Shift miniature",
                    "Leading lines", "Deep focus", "Candid snapshot",
"Aerial view",
                    "Worm's-eye view", "Character looking away",
"Panoramic landscape", "Macro detail",
                    "Framing (doorway, window)", "Triptych layout",
"Negative space emphasis", "Over-the-shoulder shot",
                    "Symmetry", "Asymmetry", "Rule of Odds",
"Framing",
                    "Negative Space", "Golden Spiral", "Triangle
Composition", "Dynamic Diagonal",
                    "Vertical Format", "Horizontal Format", "Close-up
Shot", "Extreme Wide Shot",
                    "Point of View (POV)", "Bird's-Eye View", "Low
Angle Shot", "Mid-Shot",
                    "Full Body Shot", "Action Shot", "Frozen Moment",
"Foreground Focus",
                    "Background Blur", "Juxtaposition", "Repetition",
"Pattern Break",
                    "Silhouette Composition", "Centered Subject",
"Off-Center", "Shallow Depth",
                    "Deep Focus", "Fibonacci Sequence"
                ]
            }
```

```
        };


        const ASPECT_RATIOS = [
            { id: '1:1', ratio: '1:1', dimensions: '(Square)',
default: true },
            { id: '4:3', ratio: '4:3', dimensions: '(Classic)',
default: false },
            { id: '3:4', ratio: '3:4', dimensions: '(Portrait)',
default: false },
            { id: '16:9', ratio: '16:9', dimensions: '(Widescreen)',
default: false },
            { id: '9:16', ratio: '9:16', dimensions: '(Mobile Story)',
default: false }
        ];

        let selectedAspectRatio = ASPECT_RATIOS.find(r =>
r.default).id;

        // --- DOM Elements ---
        const basePromptEl = document.getElementById('basePrompt');
        const generateBtn = document.getElementById('generateBtn');
        const uploadPromptBtn =
document.getElementById('uploadPromptBtn');
        const generatedImageEl =
document.getElementById('generatedImage');
        const downloadBtn = document.getElementById('downloadBtn');
        const placeholderTextEl =
document.getElementById('placeholderText');
        const loadingIndicatorEl =
document.getElementById('loadingIndicator');
        const loadingTipEl = document.getElementById('loadingTip');
        const statusMessageEl =
document.getElementById('statusMessage');
        const categoryContainer =
document.getElementById('categoryContainer');
        const aspectRatioContainer =
document.getElementById('aspectRatioContainer');
        const imageUploadEl = document.getElementById('imageUpload');
        const loadingCategoriesEl =
document.getElementById('loadingCategories');
        const promptSuggestionsEl =
document.getElementById('promptSuggestions');
        // New Expander Toggle
        const promptExpanderToggle =
document.getElementById('promptExpanderToggle');
```

```
        // --- Utility Functions ---

        /**
         * Custom Exponential Backoff Fetch function for API calls.
         */
        async function customFetch(url, options, maxRetries = 3) {
            for (let i = 0; i < maxRetries; i++) {
                try {
                    const response = await fetch(url, options);
                    if (response.status === 429 && i < maxRetries - 1)
{
                        const delay = Math.pow(2, i) * 1000 +
Math.random() * 1000;
                        await new Promise(resolve =>
setTimeout(resolve, delay));
                        continue;
                    }
                    if (!response.ok) {
                        const errorBody = await response.json();
                        throw new Error(`API Error ${response.status}:
${errorBody.error?.message || 'Unknown error'}`);
                    }
                    return response;
                } catch (error) {
                    if (i === maxRetries - 1) throw error;
                }
            }
        }

        /**
         * Converts a File object to a Base64 encoded string.
         */
        function fileToBase64(file) {
            return new Promise((resolve, reject) => {
                const reader = new FileReader();
                reader.onload = () =>
resolve(reader.result.split(',')[1]);
                reader.onerror = error => reject(error);
                reader.readAsDataURL(file);
            });
        }

        /**
         * Displays status/error messages.
         * @param {string} type - 'error' or 'success'
         * @param {string} message - The message content.
         * @param {number} [duration=15000] - Duration in milliseconds
before auto-hiding. Use 0 for no auto-hide.
```

```
         */
        function showStatus(type, message, duration = 15000) {
            statusMessageEl.innerHTML = message;

            // Reset classes
            statusMessageEl.classList.remove('hidden', 'status-error',
'status-success');

            // Set type class
            if (type === 'error') {
                statusMessageEl.classList.add('status-error');
            } else {
                statusMessageEl.classList.add('status-success');
            }

            // Hide download button on error
            if (type === 'error') {
                 downloadBtn.classList.add('hidden');
            }

            // Set duration for auto-hide
            if (duration > 0) {
                clearTimeout(statusMessageEl.timeoutId);
                statusMessageEl.timeoutId = setTimeout(() => {
                    statusMessageEl.classList.add('hidden');
                }, duration);
            }
        }

        /**
         * Converts PNG Base64 data (as typically returned by the API)
to JPEG Base64 data.
         * @param {string} base64PngData - Base64 string of the PNG
image.
         * @returns {Promise<string>} Base64 string of the JPEG image.
         */
        function convertBase64ToJpg(base64PngData) {
            return new Promise((resolve) => {
                const img = new Image();
                // Set the PNG data URL as the image source
                img.src = `${base64PngData}`;

                img.onload = () => {
                    const canvas = document.createElement('canvas');
                    canvas.width = img.width;
                    canvas.height = img.height;
                    const ctx = canvas.getContext('2d');
                    // Ensure black background for transparency
```

```
conversion, if any (matches dark mode theme)
                    ctx.fillStyle = '#0d1117';
                    ctx.fillRect(0, 0, canvas.width, canvas.height);
                    ctx.drawImage(img, 0, 0);

                    // Convert canvas content to JPEG base64 string
(0.9 is quality)
                    const jpgUrl = canvas.toDataURL('image/jpeg',
0.9);

                    // Extract just the base64 part
                    resolve(jpgUrl.split(',')[1]);
                };
                img.onerror = () => {
                    // Fallback in case image loading fails, just
return the original data.
                    console.error("Failed to load image for JPG
conversion. Returning original data.");
                    resolve(base64PngData);
                };
            });
        }


        // --- Firestore Functions ---

        /**
         * Seeds the Firestore database with the initial rich category
data.
         */
        async function seedDatabase() {
            // Data is saved to private path:
/artifacts/{appId}/users/{userId}/prompt_modifiers/options
            const modifiersDocRef = doc(db,
`artifacts/${appId}/users/${userId}/prompt_modifiers/options`);

            try {
                const docSnap = await getDoc(modifiersDocRef);

                if (!docSnap.exists()) {
                    await setDoc(modifiersDocRef, SEED_DATA);
                    console.log("Database seeded successfully with
initial modifier data.");
                    showStatus('success', 'Modifier options loaded and
saved to your private database for scaling!', 5000);
                } else {
                    console.log("Modifier data already exists.
Skipping seed.");
```

```
                }
            } catch (error) {
                console.error("Error seeding database:", error);
                showStatus('error', 'Could not save initial modifier
data to the database.', 10000);
            }
        }


        /**
         * Sets up real-time listener for category data from
Firestore.
         */
        function subscribeToModifiers() {
            const modifiersDocRef = doc(db,
`artifacts/${appId}/users/${userId}/prompt_modifiers/options`);

            onSnapshot(modifiersDocRef, (docSnap) => {
                if (docSnap.exists()) {
                    CATEGORY_DATA = docSnap.data();
                    renderCategories();
                    loadingCategoriesEl.classList.add('hidden');
                } else {
                    console.log("No modifier data found. Seeding
database.");
                    seedDatabase();
                }
            }, (error) => {
                console.error("Error loading modifiers:", error);
                showStatus('error', 'Failed to load dynamic modifiers.
Check console for details.', 10000);
                loadingCategoriesEl.textContent = 'Failed to load
modifiers.';
            });
        }

        /**
         * Saves a new modifier option to Firestore and updates the
UI.
         * @param {string} categoryKey - The key of the category to
update ('artist', 'style', etc.)
         */
        window.addModifier = async (categoryKey) => {
            const inputEl =
document.getElementById(`new-modifier-${categoryKey}`);
            let newValue = inputEl.value.trim();

            if (!newValue) {
```

```
                showStatus('error', 'Please enter a value before
trying to add a new modifier.', 3000);
                return;
            }

            // Capitalize first letter for consistency
            newValue = newValue.charAt(0).toUpperCase() +
newValue.slice(1);

            const currentOptions = CATEGORY_DATA[categoryKey]?.options
|| [];

            if (currentOptions.includes(newValue)) {
                showStatus('error', `"${newValue}" is already in the
list for this category.`, 3000);
                inputEl.value = '';
                return;
            }

            const modifiersDocRef = doc(db,
`artifacts/${appId}/users/${userId}/prompt_modifiers/options`);

            try {
                // Update the array field within the specific category
object
                const updatePayload = {};
                // Use a temporary array to build the new list
(Firestore array operations aren't ideal here)
                const newOptionsList = [...currentOptions,
newValue].sort();
                updatePayload[categoryKey] = {
                    ...CATEGORY_DATA[categoryKey],
                    options: newOptionsList
                };

                // Use updateDoc to merge the changes without
overwriting other categories
                await updateDoc(modifiersDocRef, updatePayload);

                // Clear input and show success message
                inputEl.value = '';
                showStatus('success', `"${newValue}" added and
permanently saved to your custom list!`, 4000);

            } catch (error) {
                console.error("Error adding modifier:", error);
                showStatus('error', `Failed to save new modifier:
${error.message}`, 10000);
```

```
            }
        };


        /**
         * Initializes Firebase and authenticates the user.
         */
        async function initFirebaseAndLoadData() {
            if (!firebaseConfig.apiKey) {
                console.error("Firebase config missing. Cannot
initialize Firestore.");
                loadingCategoriesEl.textContent = 'Database setup
error.';
                return;
            }

            try {
                const app = initializeApp(firebaseConfig);
                db = getFirestore(app);
                auth = getAuth(app);
                setLogLevel('debug');

                await setPersistence(auth, browserLocalPersistence);

                // Authentication using provided token or anonymous
sign-in
                if (initialAuthToken) {
                    const userCredential = await
signInWithCustomToken(auth, initialAuthToken);
                    userId = userCredential.user.uid;
                } else {
                    const userCredential = await
signInAnonymously(auth);
                    userId = userCredential.user.uid;
                }

                console.log("Firebase initialized. User ID:", userId);

                // Load Modifiers from DB
                subscribeToModifiers();

                // Initialize Prompts
                refreshPrompts();

            } catch (error) {
                console.error("Firebase authentication failed:",
error);
                showStatus('error', 'Authentication failed. Data
```

```
loading stopped.', 10000);
                loadingCategoriesEl.textContent = 'Authentication
error.';
            }
        }

        // --- Prompt Generation and Rendering ---

        /**
         * Selects a specified number of random prompts from the full
library.
         * @param {number} count - The number of prompts to select.
         * @returns {string[]} An array of randomly selected prompts.
         */
        function getRandomPrompts(count) {
            const shuffled = [...FULL_PROMPT_LIBRARY]
                .sort(() => 0.5 - Math.random());
            return shuffled.slice(0, count);
        }

        /**
         * Refreshes the list of displayed prompt suggestions.
         */
        window.refreshPrompts = () => {
            currentSamplePrompts = getRandomPrompts(4); // Select 4
random prompts
            renderPromptSuggestions();
            showStatus('success', 'New creative ideas loaded!', 3000);
        };

        /**
         * Renders clickable prompt suggestions based on
`currentSamplePrompts`.
         */
        function renderPromptSuggestions() {
            promptSuggestionsEl.innerHTML =
currentSamplePrompts.map((prompt, index) => `
                <div class="p-3 bg-[#0d1117] rounded-lg border
border-[#30363d] cursor-pointer hover:border-blue-500 transition
duration-150"

onclick="applyPromptSuggestion('${prompt.replace(/'/g, "\\'")}')">
                    <p class="text-sm text-gray-400
truncate">${prompt}</p>
                </div>
            `).join('');
        }
```

```
        /**
         * Applies a selected prompt suggestion to the main textarea.
         */
        window.applyPromptSuggestion = (promptText) => {
            basePromptEl.value = promptText;
            showStatus('success', 'Prompt loaded! Now adjust the
aspect ratio and click GENERATE AI ART.', 5000);
        };


        // --- Initializers & UI Renderers ---

        /**
         * Renders the aspect ratio buttons.
         */
        function renderAspectRatios() {
            aspectRatioContainer.innerHTML = ASPECT_RATIOS.map(ratio
=> `
                <button
                    id="ratio-${ratio.id.replace(':', '-')}"
                    data-ratio="${ratio.id}"
                    class="p-2 text-sm rounded-lg border transition
duration-150 ${ratio.default ? 'bg-blue-600 border-blue-600
text-white' : 'bg-[#30363d] border-[#30363d] text-gray-300
hover:bg-blue-500 hover:border-blue-500'}"
                    onclick="selectAspectRatio('${ratio.id}')">
                    ${ratio.ratio} ${ratio.dimensions}
                </button>
            `).join('');
        }
        window.selectAspectRatio = (ratioId) => {
            selectedAspectRatio = ratioId;
            document.querySelectorAll('#aspectRatioContainer
button').forEach(btn => {
                btn.classList.remove('bg-blue-600', 'border-blue-600',
'text-white');
                btn.classList.add('bg-[#30363d]', 'border-[#30363d]',
'text-gray-300');
            });
            document.getElementById(`ratio-${ratioId.replace(':',
'-')}`).classList.add('bg-blue-600', 'border-blue-600', 'text-white');
            document.getElementById(`ratio-${ratioId.replace(':',
'-')}`).classList.remove('bg-[#30363d]', 'border-[#30363d]',
'text-gray-300');
        };

        /**
         * Renders the category pill selectors using data loaded into
```

```
CATEGORY_DATA, including the new add feature.
         */
        function renderCategories() {
            categoryContainer.innerHTML =
Object.entries(CATEGORY_DATA).map(([key, data]) => {
                const optionCount = Array.isArray(data.options) ?
data.options.length : 0;
                return `
                    <div class="space-y-3">
                        <h3 class="text-lg font-medium
text-white">${data.title} (${optionCount} options)</h3>
                        <div id="pills-${key}"
class="scrollable-category flex flex-wrap gap-2 p-2 bg-[#0d1117]
rounded-lg border border-[#30363d]">
                            ${Array.isArray(data.options) ?
data.options.sort().map(option => `
                                <span class="pill inline-flex
items-center px-3 py-1 text-sm font-medium bg-[#30363d] text-gray-300
rounded-full border border-transparent ${selectedOptions[key] &&
selectedOptions[key].has(option) ? 'selected' : ''}"
                                    data-category="${key}"
                                    data-value="${option}"
                                    onclick="togglePill(this,
'${key}', '${option.replace(/'/g, "\\'")}')">
                                    ${option}
                                </span>
                            `).join('') : '<p class="text-gray-500">No
options loaded.</p>'}
                        </div>

                        <div class="flex gap-2 pt-2 border-t
border-gray-700">
                            <input type="text"
id="new-modifier-${key}" placeholder="Type and Add Custom Option"
class="flex-grow p-2 text-sm bg-[#0d1117] border border-[#30363d]
rounded-lg text-white"
                                onkeydown="if(event.key === 'Enter')
addModifier('${key}')">
                            <button onclick="addModifier('${key}')"
class="bg-blue-700 hover:bg-blue-800 text-white font-semibold py-2
px-4 text-sm rounded-lg transition duration-200">
                                Add
                            </button>
                        </div>
                    </div>
                `;
            }).join('');
        }
```

```
        window.togglePill = (element, category, value) => {
            if (!selectedOptions[category]) {
                selectedOptions[category] = new Set();
            }

            // Exclusive selection for these categories to prevent
overly long prompts
            if (category === 'artist' || category === 'style' ||
category === 'lighting') {
                // Clear all others in the category
                const container =
document.getElementById(`pills-${category}`);

container.querySelectorAll('.pill.selected').forEach(el => {
                    if (el !== element) { // Don't deselect the
element if it was already selected
                        el.classList.remove('selected');
                    }
                });
                selectedOptions[category].clear();

                // If the current element was clicked, it becomes the
new selection
                if (!element.classList.contains('selected')) {
                    element.classList.add('selected');
                    selectedOptions[category].add(value);
                } else {
                    // If the user clicks an already selected
exclusive pill, deselect it
                    element.classList.remove('selected');
                    selectedOptions[category].delete(value);
                }
            } else {
                // Toggle logic for non-exclusive categories
                if (element.classList.contains('selected')) {
                    element.classList.remove('selected');
                    selectedOptions[category].delete(value);
                } else {
                    element.classList.add('selected');
                    selectedOptions[category].add(value);
                }
            }
        };

        // Initialize UI components
        renderAspectRatios();
        initFirebaseAndLoadData();
```

```
        // --- Core Prompt Functions ---

        /**
         * Compiles the final detailed prompt from all inputs.
         * @param {boolean} includeQuality - Whether to append generic
quality modifiers. Only true if NOT expanding.
         * @returns {string} The complete prompt string.
         */
        function compilePrompt(includeQuality = true) {
            let parts = [basePromptEl.value.trim()];

            // 1. Add category modifiers
            Object.entries(CATEGORY_DATA).forEach(([key, data]) => {
                if (selectedOptions[key] && selectedOptions[key].size
> 0) {
                    const modifiers =
Array.from(selectedOptions[key]).join(', ');
                    parts.push(`${data.prefix} ${modifiers}`);
                }
            });

            // 2. Add technical quality modifiers ONLY if expansion is
NOT used.
            if (includeQuality) {
                parts.push("ultra quality, professional photography,
masterpiece, sharp focus, 8k, photorealistic");
            }

            return parts.filter(p => p).join(', ');
        }

        /**
         * Uses Gemini to expand a simple prompt into a highly
detailed, professional-grade prompt.
         * @param {string} simplePrompt - The prompt compiled from
user input and selected modifiers.
         * @returns {Promise<string>} The highly detailed, expanded
prompt.
         */
        async function expandPromptWithGemini(simplePrompt) {
            const systemPrompt = `You are an expert AI image prompt
engineer specializing in cinematic, hyper-detailed, and technically
perfect descriptions for models like Imagen.
            Your task is to take the user's prompt (which includes
style and subject keywords) and expand it into a single, cohesive,
long paragraph that is highly complex, precise, and descriptive.
```

```
            Crucially, you must:
            1. Elaborate significantly on the subject, setting, and
mood using vivid adjectives and scene details.
            2. Incorporate specific, advanced technical terms related
to professional photography, 3D rendering, and visual quality (e.g.,
Canon EOS R5, 85mm f/1.4, cinematic film grain, volumetric lighting,
Octane render, RTX Global Illumination, ultra-detailed texture, depth
of field, photorealistic, Rule of Thirds).
            3. Ensure the output is *only* the new, expanded prompt
string. Do not include introductory phrases, explanations, or
quotes.`;

            const payload = {
                contents: [{ parts: [{ text: simplePrompt }] }],
                systemInstruction: { parts: [{ text: systemPrompt }]
},
                generationConfig: {
                    temperature: 0.3 // Use low temperature for
precision
                }
            };

            const response = await customFetch(EXPANDER_URL, {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify(payload)
            });

            const result = await response.json();
            const expandedText =
result.candidates?.[0]?.content?.parts?.[0]?.text;

            if (!expandedText || expandedText.trim() === '') {
                throw new Error("Prompt expansion failed: Received
empty response from the AI.");
            }

            return expandedText.trim();
        }

        /**
         * Executes the image generation API call.
         */
        async function generateArt() {
            if (!basePromptEl.value.trim()) {
                showStatus('error', 'Please enter a base concept
prompt.', 5000);
                return;
```

```
            }

            const shouldExpand = promptExpanderToggle.checked;
            // Only include quality keywords if NOT expanding, as the
LLM will add them otherwise.
            let finalPrompt = compilePrompt(!shouldExpand);

            // UI State: Loading
            generatedImageEl.classList.add('hidden');
            downloadBtn.classList.add('hidden'); // Hide download
button while generating
            placeholderTextEl.classList.add('hidden');
            loadingIndicatorEl.classList.remove('hidden');
            statusMessageEl.classList.add('hidden');
            generateBtn.disabled = true;
            generateBtn.textContent = 'Preparing Art Generation...';
            lastGeneratedJpgBase64Data = null; // Clear previous data

            try {
                // --- STEP 1: PROMPT EXPANSION ---
                if (shouldExpand) {
                    loadingTipEl.textContent = "Step 1/2: Prompt
Expander is active. The AI is transforming your simple prompt into a
professional, hyper-detailed masterpiece...";
                    try {
                        finalPrompt = await
expandPromptWithGemini(finalPrompt);
                        console.log("Expanded Prompt:", finalPrompt);
                    } catch (error) {
                        console.error("Prompt Expansion Error:",
error);
                        showStatus('error', `Prompt expansion failed:
${error.message}. Proceeding with the original prompt.`, 7000);
                        // If expansion fails, proceed with the
original, compiled prompt.
                    }
                }


                // --- STEP 2: IMAGE GENERATION ---
                loadingTipEl.textContent = "Step 2/2: Generating
Art... Using the detailed prompt to create your image. (This may take
a moment)";

                const payload = {
                    instances: [
                        { prompt: finalPrompt }
                    ],
```

```
                    parameters: {
                        sampleCount: 1,
                        aspectRatio: selectedAspectRatio,
                    }
                };

                const response = await customFetch(GENERATE_ART_URL, {
                    method: 'POST',
                    headers: { 'Content-Type': 'application/json' },
                    body: JSON.stringify(payload)
                });

                const result = await response.json();
                const base64PngData =
result?.predictions?.[0]?.bytesBase64Encoded;

                if (!base64PngData) {
                    throw new Error("No image data returned from the
API.");
                }

                // Convert the returned PNG Base64 to JPEG Base64
                const jpgBase64Data = await
convertBase64ToJpg(base64PngData);
                lastGeneratedJpgBase64Data = jpgBase64Data; // Store
for the download button

                const imageUrl =
`${jpgBase64Data}`; // Use JPEG MIME type
                generatedImageEl.src = imageUrl;
                generatedImageEl.classList.remove('hidden');
                generatedImageEl.style.width = '100%';
                generatedImageEl.style.height = 'auto';

                const saveInstructions = `
                    <p class="text-xl font-bold mb-2">🎉 Art Generated
Successfully (No Watermark)!</p>
                    <p class="font-bold text-green-300 text-lg">---
SAVE INSTRUCTIONS (Check Downloads) ---</p>
                    <ul class="list-disc list-inside ml-4 mt-1
space-y-1">
                        <li>1. Click the **DOWNLOAD IMAGE** button
below.</li>
                        <li>2. Your browser might save the file
automatically. **Check your notifications or the Downloads folder** on
your device!</li>
                    </ul>
                `;
```

```
                showStatus('success', saveInstructions, 0); // Keep
this message visible permanently
                downloadBtn.classList.remove('hidden');

            } catch (error) {
                console.error("Art Generation Error:", error);
                showStatus('error', `Generation failed:
${error.message}. Please check your prompt and try again.`, 10000);
                generatedImageEl.src =
"https://placehold.co/600x600/161b22/c9d1d9?text=Generation+Failed";
                generatedImageEl.classList.remove('hidden');
                downloadBtn.classList.add('hidden');

            } finally {
                // UI State: Complete
                loadingIndicatorEl.classList.add('hidden');
                generateBtn.disabled = false;
                generateBtn.textContent = 'GENERATE AI ART';
            }
        }
        window.generateArt = generateArt; // Expose to HTML

        /**
         * Executes the Image-to-Prompt API call.
         */
        async function uploadAndGeneratePrompt() {
            const file = imageUploadEl.files[0];
            if (!file) {
                showStatus('error', 'Please select an image file to
upload.', 5000);
                return;
            }

            // UI State: Loading
            uploadPromptBtn.disabled = true;
            uploadPromptBtn.textContent = 'Analyzing Image...';
            loadingIndicatorEl.classList.remove('hidden');
            generatedImageEl.classList.add('hidden');
            placeholderTextEl.classList.add('hidden');
            statusMessageEl.classList.add('hidden');
            loadingTipEl.textContent = "Tip: The AI is analyzing
composition, style, color, and subject matter from your uploaded
image.";


            try {
                const base64ImageData = await fileToBase64(file);
```

```
                const userQuery = "Analyze the uploaded image. Act as
an expert AI prompt engineer. Generate a highly detailed,
single-paragraph prompt suitable for an image generator like
Midjourney or DALL-E, focusing on style, mood, lighting, and
composition. The output must be ONLY the prompt text, starting with
the core subject. Avoid adding any prefix like 'Prompt:'.";

                const payload = {
                    contents: [
                        {
                            role: "user",
                            parts: [
                                { text: userQuery },
                                {
                                    inlineData: {
                                        mimeType: file.type,
                                        data: base64ImageData
                                    }
                                }
                            ]
                        }
                    ],
                    systemInstruction: {
                         parts: [{ text: "You are an expert prompt
engineer. Your task is to analyze an image and produce a detailed,
artistic prompt. Do not include introductory text or explanations." }]
                    }
                };

                const response = await
customFetch(GENERATE_PROMPT_URL, {
                    method: 'POST',
                    headers: { 'Content-Type': 'application/json' },
                    body: JSON.stringify(payload)
                });

                const result = await response.json();
                const generatedText =
result.candidates?.[0]?.content?.parts?.[0]?.text;

                if (!generatedText) {
                    throw new Error("Could not generate prompt from
image.");
                }

                // Update the main prompt input
                basePromptEl.value =
generatedText.trim().replace(/^Prompt:\s**/i, '',); // Clean up any
```

```
prefix
                showStatus('success', 'Prompt successfully generated
from image! You can now generate art or refine the prompt.', 5000);

            } catch (error) {
                console.error("Image-to-Prompt Error:", error);
                showStatus('error', `Prompt generation failed:
${error.message}.`, 10000);
            } finally {
                // UI State: Complete
                loadingIndicatorEl.classList.add('hidden');
                uploadPromptBtn.disabled = false;
                uploadPromptBtn.textContent = 'Analyze Image &
Generate Prompt';
            }
        }
        window.uploadAndGeneratePrompt = uploadAndGeneratePrompt; //
Expose to HTML


        /**
         * Function to force a direct file download using the <a>
element and download attribute.
         */
        window.handleDownload = () => {
            if (lastGeneratedJpgBase64Data) {
                const imgUrl =
`${lastGeneratedJpgBase64Data}`;
                const filename = `AI_Art_Forge_${Date.now()}.jpg`; //
Generate a unique filename

                // Create a temporary anchor element
                const a = document.createElement('a');
                a.style.display = 'none';
                a.href = imgUrl;
                a.download = filename; // This attribute forces a
download dialog/save

                // Append, click, and remove the element
                document.body.appendChild(a);
                a.click();
                document.body.removeChild(a);

                showStatus('success', '📥 Download initiated! Please
check your browser notifications or downloads folder for the file.',
7000);

            } else {
```

```
                showStatus('error', 'Image data is not ready. Please
generate art first.', 7000);
            }
        };


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