<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Analogy AI</title>

<!-- Tailwind CSS -->

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

<!-- Google Fonts: Space Mono & Roboto Mono -->

<link rel="preconnect" href="https://fonts.googleapis.com">

<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>

<link href="https://fonts.googleapis.com/css2?family=Roboto+Mono:wght@400;700&family=Space+Mono:wght@400;700&display=swap" rel="stylesheet">

<style>

/\* Define custom fonts and base styles \*/

body {

background-color: #121212;

color: rgba(255, 255, 255, 0.87);

font-family: 'Roboto Mono', monospace;

overflow-x: hidden;

}

/\* Custom font classes for Tailwind \*/

.font-spacemono {

font-family: 'Space Mono', monospace;

}

.font-robotomono {

font-family: 'Roboto Mono', monospace;

}

/\* Glassmorphism Card Style \*/

.glass-card {

background: rgba(255, 255, 255, 0.05);

backdrop-filter: blur(12px);

-webkit-backdrop-filter: blur(12px); /\* Safari compatibility \*/

border-radius: 16px;

border: 1px solid rgba(255, 255, 255, 0.15);

box-shadow: 0 8px 32px 0 rgba(0, 0, 0, 0.37);

}

/\* Custom styles for inputs to match the theme \*/

.themed-input {

background: rgba(0, 0, 0, 0.2);

border: 1px solid rgba(255, 255, 255, 0.1);

color: rgba(255, 255, 255, 0.87);

transition: border-color 0.3s, box-shadow 0.3s;

}

.themed-input:focus {

outline: none;

border-color: #FFC107;

box-shadow: 0 0 0 2px rgba(255, 193, 7, 0.5);

}

/\* Animated background \*/

.background-gradient {

position: fixed;

top: 0;

left: 0;

width: 100%;

height: 100%;

z-index: -1;

background: radial-gradient(circle at 10% 20%, rgba(255, 193, 7, 0.1), transparent 30%),

radial-gradient(circle at 80% 90%, rgba(255, 193, 7, 0.1), transparent 30%);

animation: animate-gradient 20s ease-in-out infinite;

}

@keyframes animate-gradient {

0% { background-position: 0% 50%; }

50% { background-position: 100% 50%; }

100% { background-position: 0% 50%; }

}

/\* Custom scrollbar for a better aesthetic \*/

::-webkit-scrollbar {

width: 8px;

}

::-webkit-scrollbar-track {

background: #121212;

}

::-webkit-scrollbar-thumb {

background: #FFC107;

border-radius: 4px;

}

::-webkit-scrollbar-thumb:hover {

background: #FFA000;

}

/\* Loading Spinner \*/

.loader {

border: 4px solid rgba(255, 255, 255, 0.2);

border-left-color: #FFC107;

border-radius: 50%;

width: 40px;

height: 40px;

animation: spin 1s linear infinite;

}

@keyframes spin {

to { transform: rotate(360deg); }

}

</style>

</head>

<body class="antialiased">

<div class="background-gradient"></div>

<!-- Main Container -->

<div class="relative min-h-screen flex flex-col items-center justify-center p-4 md:p-8 transition-opacity duration-500" id="app-container">

<!-- Header / Navigation -->

<header class="w-full max-w-4xl mx-auto absolute top-0 left-1/2 -translate-x-1/2 p-4">

<nav class="glass-card flex justify-between items-center p-4">

<h1 class="font-spacemono text-xl md:text-2xl font-bold text-white">

Analogy<span class="text-[#FFC107]">AI</span>

</h1>

<div id="auth-container">

<!-- Auth buttons will be injected here -->

</div>

</nav>

</header>

<!-- Logged Out View -->

<main id="logged-out-view" class="w-full max-w-2xl text-center mt-20">

<div class="glass-card p-8 md:p-12">

<h2 class="font-spacemono text-3xl md:text-4xl font-bold text-white mb-4">Unlock Understanding</h2>

<p class="text-lg text-gray-300 mb-8">Translate complex topics into simple, personalized analogies. Sign in to begin your learning journey.</p>

<button id="sign-in-btn-main" class="bg-[#FFC107] text-black font-bold font-spacemono py-3 px-8 rounded-lg hover:bg-[#FFA000] transition-colors duration-300 text-lg">

Sign In with Google

</button>

</div>

</main>

<!-- Logged In View (Main App) -->

<main id="logged-in-view" class="hidden w-full max-w-4xl mx-auto mt-32 md:mt-40 space-y-8">

<!-- Input Card -->

<section id="generator-section" class="glass-card p-6 md:p-8">

<form id="analogy-form">

<h2 class="font-spacemono text-2xl font-bold text-white mb-6">Generate an Analogy</h2>

<div class="space-y-6">

<div>

<label for="topic" class="block text-sm font-medium text-gray-300 mb-2">Complex Topic</label>

<input type="text" id="topic" name="topic" class="themed-input w-full p-3 rounded-lg" placeholder="e.g., Quantum Computing, Blockchain, APIs" required>

</div>

<div>

<label for="interests" class="block text-sm font-medium text-gray-300 mb-2">Personalize with Interests</label>

<input type="text" id="interests" name="interests" class="themed-input w-full p-3 rounded-lg" placeholder="e.g., cooking, basketball, video games" required>

</div>

</div>

<div class="mt-8 text-center">

<button type="submit" id="generate-btn" class="bg-[#FFC107] text-black font-bold font-spacemono py-3 px-10 rounded-lg hover:bg-[#FFA000] transition-colors duration-300 text-lg w-full md:w-auto flex items-center justify-center gap-3">

<svg id="generate-icon" xmlns="http://www.w3.org/2000/svg" width="20" height="20" fill="currentColor" class="bi bi-stars" viewBox="0 0 16 16"><path d="M7.657 6.247c.11-.33.576-.33.686 0l.645 1.937a2.89 2.89 0 0 0 1.829 1.828l1.936.645c.33.11.33.576 0 .686l-1.937.645a2.89 2.89 0 0 0-1.828 1.829l-.645 1.936a.361.361 0 0 1-.686 0l-.645-1.937a2.89 2.89 0 0 0-1.828-1.828l-1.937-.645a.361.361 0 0 1 0-.686l1.937-.645a2.89 2.89 0 0 0 1.828-1.828zM3.794 1.148a.217.217 0 0 1 .412 0l.387 1.162c.173.518.579.924 1.097 1.097l1.162.387a.217.217 0 0 1 0 .412l-1.162.387A1.73 1.73 0 0 0 4.58 5.48l-.386 1.162a.217.217 0 0 1-.412 0l-.387-1.162A1.73 1.73 0 0 0 2.309 4.22l-1.162-.387a.217.217 0 0 1 0-.412l1.162-.387A1.73 1.73 0 0 0 3.407 2.31zM10.863.099a.145.145 0 0 1 .274 0l.258.774c.115.346.386.617.732.732l.774.258a.145.145 0 0 1 0 .274l-.774.258a1.16 1.16 0 0 0-.732.732l-.258.774a.145.145 0 0 1-.274 0l-.258-.774a1.16 1.16 0 0 0-.732-.732l-.774-.258a.145.145 0 0 1 0-.274l.774-.258c.346-.115.617-.386.732-.732z"/></svg>

<span id="generate-btn-text">Generate</span>

</button>

</div>

</form>

</section>

<!-- Loading Indicator -->

<div id="loader" class="hidden justify-center py-8">

<div class="loader"></div>

</div>

<!-- Results Section -->

<section id="results-section" class="space-y-6">

<!-- Generated analogies will be prepended here -->

</section>

<!-- History Section -->

<section id="history-section" class="space-y-4 pt-8">

<h2 class="font-spacemono text-2xl font-bold text-white mb-4">History</h2>

<div id="history-container" class="space-y-4">

<!-- History items will be populated here -->

</div>

<p id="no-history-msg" class="text-center text-gray-400 py-4 hidden">Your generated analogies will appear here.</p>

</section>

</main>

</div>

<!-- Firebase SDKs -->

<script type="module">

// Import necessary Firebase functions

import { initializeApp } from "https://www.gstatic.com/firebasejs/10.12.2/firebase-app.js";

import {

getAuth,

onAuthStateChanged,

GoogleAuthProvider,

signInWithPopup,

signOut

} from "https://www.gstatic.com/firebasejs/10.12.2/firebase-auth.js";

import {

getFirestore,

collection,

addDoc,

query,

where,

onSnapshot,

serverTimestamp,

orderBy

} from "https://www.gstatic.com/firebasejs/10.12.2/firebase-firestore.js";

// --- CONFIGURATION ---

// IMPORTANT: Replace with your actual Firebase project configuration.

const firebaseConfig = {

apiKey: "YOUR\_API\_KEY",

authDomain: "YOUR\_AUTH\_DOMAIN",

projectId: "YOUR\_PROJECT\_ID",

storageBucket: "YOUR\_STORAGE\_BUCKET",

messagingSenderId: "YOUR\_MESSAGING\_SENDER\_ID",

appId: "YOUR\_APP\_ID"

};

// IMPORTANT: Firestore security rules to set in your Firebase console.

// This ensures users can only access their own data.

/\*

rules\_version = '2';

service cloud.firestore {

match /databases/{database}/documents {

// Users can only read and write to their own profile.

match /users/{userId} {

allow read, write: if request.auth.uid == userId;

}

// Users can create analogies if the userId matches their auth uid.

// Users can only read, update, or delete their own analogies.

match /analogies/{analogyId} {

allow create: if request.resource.data.userId == request.auth.uid;

allow read, update, delete: if resource.data.userId == request.auth.uid;

}

}

}

\*/

// --- INITIALIZATION ---

const app = initializeApp(firebaseConfig);

const auth = getAuth(app);

const db = getFirestore(app);

const analogiesCollection = collection(db, "analogies");

// --- DOM ELEMENTS ---

const authContainer = document.getElementById('auth-container');

const signInBtnMain = document.getElementById('sign-in-btn-main');

const loggedOutView = document.getElementById('logged-out-view');

const loggedInView = document.getElementById('logged-in-view');

const analogyForm = document.getElementById('analogy-form');

const generateBtn = document.getElementById('generate-btn');

const generateBtnText = document.getElementById('generate-btn-text');

const generateIcon = document.getElementById('generate-icon');

const resultsSection = document.getElementById('results-section');

const historyContainer = document.getElementById('history-container');

const noHistoryMsg = document.getElementById('no-history-msg');

const loader = document.getElementById('loader');

let currentUser = null;

let unsubscribeHistory = null;

// --- AUTHENTICATION ---

const googleProvider = new GoogleAuthProvider();

const handleSignIn = async () => {

try {

await signInWithPopup(auth, googleProvider);

} catch (error) {

console.error("Error during sign in:", error);

alert("Could not sign in. Please try again.");

}

};

const handleSignOut = async () => {

try {

await signOut(auth);

} catch (error) {

console.error("Error during sign out:", error);

}

};

onAuthStateChanged(auth, (user) => {

currentUser = user;

if (user) {

// User is signed in

loggedOutView.classList.add('hidden');

loggedInView.classList.remove('hidden');

updateAuthUI(user);

loadHistory(user.uid);

} else {

// User is signed out

loggedOutView.classList.remove('hidden');

loggedInView.classList.add('hidden');

updateAuthUI(null);

if (unsubscribeHistory) {

unsubscribeHistory();

unsubscribeHistory = null;

}

historyContainer.innerHTML = '';

}

});

const updateAuthUI = (user) => {

authContainer.innerHTML = '';

if (user) {

const welcomeText = document.createElement('span');

welcomeText.className = 'text-gray-300 hidden md:inline';

welcomeText.textContent = `Welcome, ${user.displayName.split(' ')[0]}`;

const signOutBtn = document.createElement('button');

signOutBtn.textContent = 'Sign Out';

signOutBtn.className = 'bg-red-600 text-white font-bold font-spacemono py-2 px-4 rounded-lg hover:bg-red-700 transition-colors duration-300 ml-4';

signOutBtn.onclick = handleSignOut;

authContainer.appendChild(welcomeText);

authContainer.appendChild(signOutBtn);

} else {

const signInBtn = document.createElement('button');

signInBtn.textContent = 'Sign In';

signInBtn.className = 'bg-[#FFC107] text-black font-bold font-spacemono py-2 px-4 rounded-lg hover:bg-[#FFA000] transition-colors duration-300';

signInBtn.onclick = handleSignIn;

authContainer.appendChild(signInBtn);

}

};

signInBtnMain.onclick = handleSignIn;

// --- FIRESTORE & HISTORY ---

const loadHistory = (uid) => {

if (unsubscribeHistory) unsubscribeHistory();

const q = query(analogiesCollection, where("userId", "==", uid), orderBy("createdAt", "desc"));

unsubscribeHistory = onSnapshot(q, (snapshot) => {

if (snapshot.empty) {

noHistoryMsg.classList.remove('hidden');

historyContainer.innerHTML = '';

} else {

noHistoryMsg.classList.add('hidden');

historyContainer.innerHTML = '';

snapshot.forEach(doc => {

const analogyData = doc.data();

const historyCard = createAnalogyCard(analogyData, false);

historyContainer.appendChild(historyCard);

});

}

}, (error) => {

console.error("Error fetching history:", error);

alert("Could not load your history. Please refresh the page.");

});

};

const saveAnalogyToHistory = async (analogyData) => {

if (!currentUser) return;

try {

await addDoc(analogiesCollection, {

...analogyData,

userId: currentUser.uid,

createdAt: serverTimestamp()

});

} catch (error) {

console.error("Error saving to Firestore:", error);

}

};

// --- AI CORE (GEMINI API) ---

const generateAnalogy = async (topic, interests) => {

// This is a simulation of the backend logic described in the blueprint.

// In a real production app, this fetch call would be made from a secure backend (like Google Cloud Run)

// to protect the API key.

const apiKey = ""; // Leave empty for Canvas environment

const apiUrl = `https://generativelanguage.googleapis.com/v1beta/models/gemini-2.5-flash-preview-05-20:generateContent?key=${apiKey}`;

// This prompt is engineered based on the blueprint's specifications.

const prompt = `

\*\*Persona:\*\* You are "Analogy AI," an expert educator. Your mission is to explain complex topics in a simple, relatable, and personalized way. You are patient, clear, and encouraging.

\*\*Core Instruction:\*\* Generate a simple, easy-to-understand analogy for the complex topic provided. The analogy must be personalized based on the user's provided interests. After the analogy, provide one concrete, real-world example of the topic in action.

\*\*Constraints:\*\* The total response must be concise and accessible to a high school student. Do not be overly technical.

\*\*User-Provided Context:\*\*

- Complex Topic: "${topic}"

- Personalize with these interests: "${interests}"

\*\*Output Formatting Instruction:\*\* Provide your response \*exclusively\* in a valid JSON format. Do not include any other text, explanations, or markdown formatting outside of the JSON structure. The JSON object must have two keys: "analogy" and "example".

\*\*Example of desired output format:\*\*

{

"analogy": "Cloud computing is like ordering a pizza...",

"example": "A real-world example is Netflix..."

}

`;

const payload = {

contents: [{

parts: [{ text: prompt }]

}],

generationConfig: {

responseMimeType: "application/json",

responseSchema: {

type: "OBJECT",

properties: {

"analogy": { "type": "STRING" },

"example": { "type": "STRING" }

},

required: ["analogy", "example"]

}

}

};

// Exponential backoff for retries

let response;

let delay = 1000;

for (let i = 0; i < 3; i++) {

try {

response = await fetch(apiUrl, {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify(payload)

});

if (response.ok) {

const result = await response.json();

if (result.candidates && result.candidates.length > 0) {

const jsonText = result.candidates[0].content.parts[0].text;

return JSON.parse(jsonText);

}

}

} catch (error) {

console.error(`Attempt ${i + 1} failed:`, error);

}

await new Promise(res => setTimeout(res, delay));

delay \*= 2;

}

throw new Error("Failed to get a valid response from the AI model after multiple attempts.");

};

// --- UI & EVENT HANDLING ---

analogyForm.addEventListener('submit', async (e) => {

e.preventDefault();

if (!currentUser) {

alert("Please sign in to generate an analogy.");

return;

}

const topic = document.getElementById('topic').value.trim();

const interests = document.getElementById('interests').value.trim();

if (!topic || !interests) {

alert("Please fill out both fields.");

return;

}

// Start loading state

setLoading(true);

resultsSection.innerHTML = '';

try {

const result = await generateAnalogy(topic, interests);

const analogyData = { topic, interests, ...result };

const resultCard = createAnalogyCard(analogyData, true);

resultsSection.prepend(resultCard);

await saveAnalogyToHistory(analogyData);

} catch (error) {

console.error("Analogy generation failed:", error);

const errorCard = createErrorCard(error.message);

resultsSection.prepend(errorCard);

} finally {

// End loading state

setLoading(false);

analogyForm.reset();

}

});

const setLoading = (isLoading) => {

if (isLoading) {

generateBtn.disabled = true;

generateBtn.classList.add('opacity-50', 'cursor-not-allowed');

generateBtnText.textContent = 'Generating...';

generateIcon.classList.add('hidden');

loader.classList.remove('hidden');

loader.classList.add('flex');

} else {

generateBtn.disabled = false;

generateBtn.classList.remove('opacity-50', 'cursor-not-allowed');

generateBtnText.textContent = 'Generate';

generateIcon.classList.remove('hidden');

loader.classList.add('hidden');

loader.classList.remove('flex');

}

};

const createAnalogyCard = (data, isNew) => {

const card = document.createElement('div');

card.className = `glass-card p-6 md:p-8 ${isNew ? 'animate-fade-in' : ''}`;

card.innerHTML = `

<h3 class="font-spacemono text-xl font-bold text-[#FFC107] mb-1">Analogy for: ${data.topic}</h3>

<p class="text-sm text-gray-400 mb-4">Based on your interest in: ${data.interests}</p>

<div class="mt-4 border-t border-gray-700 pt-4">

<h4 class="font-spacemono text-lg font-semibold text-white mb-2">The Analogy</h4>

<p class="text-gray-300 leading-relaxed">${data.analogy}</p>

</div>

<div class="mt-6 border-t border-gray-700 pt-4">

<h4 class="font-spacemono text-lg font-semibold text-white mb-2">Real-World Example</h4>

<p class="text-gray-300 leading-relaxed">${data.example}</p>

</div>

`;

return card;

};

const createErrorCard = (message) => {

const card = document.createElement('div');

card.className = 'glass-card p-6 md:p-8 border-red-500';

card.innerHTML = `

<h3 class="font-spacemono text-xl font-bold text-red-400 mb-2">Generation Failed</h3>

<p class="text-gray-300">Sorry, we couldn't generate an analogy at this time. Please try again later.</p>

<p class="text-xs text-gray-500 mt-4">Error: ${message}</p>

`;

return card;

};

</script>

</body>

</html>