<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>AI Travel Planner</title>

    <!-- Tailwind CSS CDN for styling -->

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

    <!-- Inter font for better aesthetics -->

    <link href="https://fonts.googleapis.com/css2?family=Inter:wght@400;500;600;700;800&display=swap" rel="stylesheet">

    <style>

        body {

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

        }

    </style>

</head>

<body>

    <div id="root"></div>

    <!-- React, React DOM, and Babel for JSX transformation -->

    <script src="https://unpkg.com/react@18/umd/react.production.min.js"></script>

    <script src="https://unpkg.com/react-dom@18/umd/react-dom.production.min.js"></script>

    <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>

    <!-- Firebase SDKs -->

    <!-- This script loads Firebase modules as ES modules and makes key functions

         and Canvas-injected variables globally accessible for the subsequent Babel script. -->

    <script type="module">

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

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

        import { getFirestore, doc, getDoc, addDoc, setDoc, updateDoc, deleteDoc, onSnapshot, collection, query, where, getDocs } from "https://www.gstatic.com/firebasejs/11.6.1/firebase-firestore.js";

        // IMPORTANT: These variables are provided by the Canvas environment.

        // For local development or GitHub deployment, you would typically

        // replace these with your actual Firebase project credentials.

        // For demonstration purposes within Canvas, they are automatically injected.

        // We are making them global with a unique prefix to avoid conflicts and ensure availability.

        window.canvas\_app\_id = typeof \_\_app\_id !== 'undefined' ? \_\_app\_id : 'default-app-id';

        window.canvas\_firebase\_config = typeof \_\_firebase\_config !== 'undefined' ? JSON.parse(\_\_firebase\_config) : {};

        window.canvas\_initial\_auth\_token = typeof \_\_initial\_auth\_token !== 'undefined' ? \_\_initial\_auth\_token : null;

        // Expose Firebase functions globally with a unique prefix

        // This ensures they are accessible by the Babel-transformed React code.

        window.fb\_initializeApp = initializeApp;

        window.fb\_getAuth = getAuth;

        window.fb\_signInAnonymously = signInAnonymously;

        window.fb\_signInWithCustomToken = signInWithCustomToken;

        window.fb\_onAuthStateChanged = onAuthStateChanged;

        window.fb\_getFirestore = getFirestore;

        window.fb\_doc = doc;

        window.fb\_getDoc = getDoc;

        window.fb\_addDoc = addDoc;

        window.fb\_setDoc = setDoc;

        window.fb\_updateDoc = updateDoc;

        window.fb\_deleteDoc = deleteDoc;

        window.fb\_onSnapshot = onSnapshot;

        window.fb\_collection = collection;

        window.fb\_query = query;

        window.fb\_where = where;

        window.fb\_getDocs = getDocs;

    </script>

    <!-- Your React App Code -->

    <!-- This script uses type="text/babel" to allow JSX syntax and accesses

         the globally exposed Firebase functions and Canvas variables. -->

    <script type="text/babel">

        // Destructure useState and useEffect from React

        const { useState, useEffect } = React;

        // Access Canvas-injected global variables

        const appId = window.canvas\_app\_id;

        const firebaseConfig = window.canvas\_firebase\_config;

        const initialAuthToken = window.canvas\_initial\_auth\_token;

        // Access Firebase functions that were exposed globally by the module script

        const initializeApp = window.fb\_initializeApp;

        const getAuth = window.fb\_getAuth;

        const signInAnonymously = window.fb\_signInAnonymously;

        const signInWithCustomToken = window.fb\_signInWithCustomToken;

        const onAuthStateChanged = window.fb\_onAuthStateChanged;

        const getFirestore = window.fb\_getFirestore;

        const doc = window.fb\_doc;

        const getDoc = window.fb\_getDoc;

        const addDoc = window.fb\_addDoc;

        const setDoc = window.fb\_setDoc;

        const updateDoc = window.fb\_updateDoc;

        const deleteDoc = window.fb\_deleteDoc;

        const onSnapshot = window.fb\_onSnapshot;

        const collection = window.fb\_collection;

        const query = window.fb\_query;

        const where = window.fb\_where;

        const getDocs = window.fb\_getDocs;

        // Main App Component

        function App() {

            const [db, setDb] = useState(null);

            const [auth, setAuth] = useState(null);

            const [userId, setUserId] = useState(null);

            const [isAuthReady, setIsAuthReady] = useState(false);

            const [destination, setDestination] = useState('');

            const [interests, setInterests] = useState('');

            const [budget, setBudget] = useState('');

            const [season, setSeason] = useState('');

            const [tripPlan, setTripPlan] = useState('');

            const [loading, setLoading] = useState(false);

            const [error, setError] = useState('');

            const [savedItineraries, setSavedItineraries] = useState([]);

            const [message, setMessage] = useState(''); // For user messages like "Plan saved!"

            // Initialize Firebase and handle authentication

            useEffect(() => {

                try {

                    const app = initializeApp(firebaseConfig);

                    const firestore = getFirestore(app);

                    const firebaseAuth = getAuth(app);

                    setDb(firestore);

                    setAuth(firebaseAuth);

                    onAuthStateChanged(firebaseAuth, async (user) => {

                        if (user) {

                            setUserId(user.uid);

                        } else {

                            // If no user, try signing in with custom token or anonymously

                            try {

                                if (initialAuthToken) {

                                    await signInWithCustomToken(firebaseAuth, initialAuthToken);

                                } else {

                                    await signInAnonymously(firebaseAuth);

                                }

                                // Fallback if still no uid, although Firebase usually provides one

                                setUserId(firebaseAuth.currentUser?.uid || crypto.randomUUID());

                            } catch (signInError) {

                                console.error("Firebase sign-in error:", signInError);

                                // Use a random ID if sign-in fails to allow local usage without full auth

                                setUserId(crypto.randomUUID());

                            }

                        }

                        setIsAuthReady(true); // Auth state determined

                    });

                } catch (e) {

                    console.error("Failed to initialize Firebase:", e);

                    setError("Failed to initialize the app. Please try again.");

                }

            }, []);

            // Fetch saved itineraries when auth is ready and user ID is available

            useEffect(() => {

                if (isAuthReady && db && userId) {

                    // Define the collection reference for public itineraries

                    const itinerariesCollectionRef = collection(db, `artifacts/${appId}/public/data/itineraries`);

                    // Create a query to filter itineraries by the current user's ID

                    // Note: Firestore queries without orderBy on a field used in `where`

                    // might still require composite indexes for complex queries in production.

                    // For this simple case, sorting is done in memory after fetching.

                    const q = query(itinerariesCollectionRef, where("userId", "==", userId));

                    // Set up a real-time listener for changes in the itineraries collection

                    const unsubscribe = onSnapshot(q, (snapshot) => {

                        const plans = snapshot.docs.map(doc => ({ id: doc.id, ...doc.data() }));

                        // Sort plans by timestamp in descending order to show most recent first

                        plans.sort((a, b) => (b.timestamp?.toDate() || 0) - (a.timestamp?.toDate() || 0));

                        setSavedItineraries(plans);

                    }, (err) => {

                        console.error("Error fetching itineraries:", err);

                        setError("Could not load saved itineraries.");

                    });

                    return () => unsubscribe(); // Cleanup the listener when component unmounts or dependencies change

                }

            }, [isAuthReady, db, userId, appId]); // Added appId to dependencies for completeness

            // Function to display a temporary message to the user

            const showMessage = (msg) => {

                setMessage(msg);

                setTimeout(() => setMessage(''), 3000); // Clear message after 3 seconds

            };

            // Handle trip plan generation

            const handleGeneratePlan = async () => {

                // Check if Firebase database and user ID are available

                if (!db || !userId) {

                    setError("App is not ready. Please wait a moment.");

                    return;

                }

                setLoading(true); // Set loading state to true

                setError(''); // Clear any previous errors

                setTripPlan(''); // Clear any previous trip plan

                // Construct the prompt for the AI model based on user inputs

                const prompt = `Generate a personalized travel itinerary for the following preferences:

- Destination: ${destination || 'any'}.

- Interests: ${interests || 'general sightseeing'}.

- Budget: ${budget || 'moderate'}.

- Season/Time: ${season || 'any time of year'}.

Please provide a detailed, engaging, and realistic itinerary. Include suggestions for activities, food, and a general timeline for a few days (e.g., 3-5 days).`;

                try {

                    let chatHistory = [];

                    chatHistory.push({ role: "user", parts: [{ text: prompt }] });

                    const payload = { contents: chatHistory };

                    const apiKey = ""; // Canvas will inject API key at runtime

                    // Define the API URL for the Gemini 2.0 Flash model

                    const apiUrl = `https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent?key=${apiKey}`;

                    // Make a POST request to the Gemini API

                    const response = await fetch(apiUrl, {

                        method: 'POST',

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

                        body: JSON.stringify(payload)

                    });

                    // Check if the API response was successful

                    if (!response.ok) {

                        throw new Error(`API error: ${response.status} ${response.statusText}`);

                    }

                    const result = await response.json(); // Parse the JSON response

                    // Check if the AI generated content and set the trip plan

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

                        result.candidates[0].content && result.candidates[0].content.parts &&

                        result.candidates[0].content.parts.length > 0) {

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

                        setTripPlan(text);

                        // Save the generated itinerary to Firestore

                        const itinerariesCollectionRef = collection(db, `artifacts/${appId}/public/data/itineraries`);

                        await addDoc(itinerariesCollectionRef, {

                            userId: userId,

                            destination,

                            interests,

                            budget,

                            season,

                            itinerary: text,

                            timestamp: new Date() // Store the current timestamp

                        });

                        showMessage("Trip plan generated and saved!");

                    } else {

                        throw new Error("No content received from AI.");

                    }

                } catch (err) {

                    console.error("Error generating trip plan:", err);

                    setError(`Failed to generate trip plan: ${err.message}. Please try again.`);

                } finally {

                    setLoading(false); // Reset loading state

                }

            };

            // Function to load a saved plan into the input fields and display area

            const loadSavedPlan = (plan) => {

                setDestination(plan.destination || '');

                setInterests(plan.interests || '');

                setBudget(plan.budget || '');

                setSeason(plan.season || '');

                setTripPlan(plan.itinerary || '');

                showMessage("Loaded saved plan.");

            };

            // Function to delete a saved plan from Firestore

            const deleteSavedPlan = async (planId) => {

                // Check if database and user ID are available

                if (!db || !userId) {

                    setError("App is not ready. Cannot delete plan.");

                    return;

                }

                try {

                    // Delete the document from the 'itineraries' collection

                    await deleteDoc(doc(db, `artifacts/${appId}/public/data/itineraries`, planId));

                    showMessage("Plan deleted successfully!");

                } catch (err) {

                    console.error("Error deleting plan:", err);

                    setError("Failed to delete plan.");

                }

            };

            // Show loading screen while authentication is in progress

            if (!isAuthReady) {

                return (

                    <div className="flex items-center justify-center min-h-screen bg-gray-100 p-4">

                        <p className="text-xl text-gray-700">Initializing app and authenticating...</p>

                    </div>

                );

            }

            // Main application UI

            return (

                <div className="min-h-screen bg-gradient-to-br from-blue-100 to-indigo-200 p-4 font-inter">

                    <div className="max-w-4xl mx-auto bg-white shadow-2xl rounded-xl p-6 md:p-8 mt-8">

                        <h1 className="text-4xl font-extrabold text-center text-gray-800 mb-6 tracking-tight">

                            <span role="img" aria-label="airplane">✈️</span>

                            AI Travel Planner

                        </h1>

                        <p className="text-center text-gray-600 mb-8 text-lg">

                            Your personalized journey starts here. Fill in your preferences and let AI craft your perfect trip!

                        </p>

                        {userId && (

                            <p className="text-sm text-center text-gray-500 mb-4">

                                Your User ID: <span className="font-mono text-blue-600">{userId}</span>

                            </p>

                        )}

                        {message && (

                            <div className="bg-green-100 text-green-700 p-3 rounded-lg mb-4 text-center border border-green-200">

                                {message}

                            </div>

                        )}

                        <div className="grid grid-cols-1 md:grid-cols-2 gap-6 mb-8">

                            <div className="space-y-4">

                                <label htmlFor="destination" className="block text-gray-700 font-semibold text-lg">Destination</label>

                                <input

                                    type="text"

                                    id="destination"

                                    className="w-full p-3 border border-gray-300 rounded-lg focus:ring-blue-500 focus:border-blue-500 shadow-sm"

                                    placeholder="e.g., Paris, Kyoto, Grand Canyon"

                                    value={destination}

                                    onChange={(e) => setDestination(e.target.value)}

                                />

                            </div>

                            <div className="space-y-4">

                                <label htmlFor="interests" className="block text-gray-700 font-semibold text-lg">Interests</label>

                                <input

                                    type="text"

                                    id="interests"

                                    className="w-full p-3 border border-gray-300 rounded-lg focus:ring-blue-500 focus:border-blue-500 shadow-sm"

                                    placeholder="e.g., historical sites, beaches, hiking, food"

                                    value={interests}

                                    onChange={(e) => setInterests(e.target.value)}

                                />

                            </div>

                            <div className="space-y-4">

                                <label htmlFor="budget" className="block text-gray-700 font-semibold text-lg">Budget</label>

                                <select

                                    id="budget"

                                    className="w-full p-3 border border-gray-300 rounded-lg focus:ring-blue-500 focus:border-blue-500 shadow-sm"

                                    value={budget}

                                    onChange={(e) => setBudget(e.target.value)}

                                >

                                    <option value="">Select your budget</option>

                                    <option value="low">Low</option>

                                    <option value="moderate">Moderate</option>

                                    <option value="high">High</option>

                                    <option value="luxury">Luxury</option>

                                </select>

                            </div>

                            <div className="space-y-4">

                                <label htmlFor="season" className="block text-gray-700 font-semibold text-lg">Ideal Season</label>

                                <select

                                    id="season"

                                    className="w-full p-3 border border-gray-300 rounded-lg focus:ring-blue-500 focus:border-blue-500 shadow-sm"

                                    value={season}

                                    onChange={(e) => setSeason(e.target.value)}

                                >

                                    <option value="">Select ideal season</option>

                                    <option value="spring">Spring</option>

                                    <option value="summer">Summer</option>

                                    <option value="autumn">Autumn</option>

                                    <option value="winter">Winter</option>

                                    <option value="any">Any time of year</option>

                                </select>

                            </div>

                        </div>

                        <button

                            onClick={handleGeneratePlan}

                            className={`w-full p-4 rounded-xl text-white font-bold text-xl shadow-lg transform transition-transform duration-200 ${

                                loading

                                    ? 'bg-blue-300 cursor-not-allowed'

                                    : 'bg-blue-600 hover:bg-blue-700 hover:scale-105 active:scale-95'

                            }`}

                            disabled={loading}

                        >

                            {loading ? (

                                <span className="flex items-center justify-center">

                                    <svg className="animate-spin -ml-1 mr-3 h-5 w-5 text-white" xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24">

                                        <circle className="opacity-25" cx="12" cy="12" r="10" stroke="currentColor" strokeWidth="4"></circle>

                                        <path className="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>

                                    Generating Plan...

                                </span>

                            ) : (

                                'Generate Personalized Trip Plan'

                            )}

                        </button>

                        {error && (

                            <div className="bg-red-100 text-red-700 p-4 rounded-lg mt-6 text-center border border-red-200">

                                <p className="font-semibold">Error:</p>

                                <p>{error}</p>

                            </div>

                        )}

                        {tripPlan && (

                            <div className="mt-8 bg-gray-50 p-6 rounded-xl shadow-inner border border-gray-200">

                                <h2 className="text-3xl font-bold text-gray-800 mb-4 text-center">Your Custom Trip Itinerary</h2>

                                <div className="prose max-w-none text-gray-700 leading-relaxed" style={{ whiteSpace: 'pre-wrap' }}>

                                    {tripPlan}

                                </div>

                            </div>

                        )}

                        {savedItineraries.length > 0 && (

                            <div className="mt-12">

                                <h2 className="text-3xl font-bold text-gray-800 mb-6 text-center">Your Saved Itineraries</h2>

                                <div className="grid grid-cols-1 gap-6">

                                    {savedItineraries.map((plan) => (

                                        <div key={plan.id} className="bg-white p-5 rounded-lg shadow-md border border-gray-200 hover:shadow-lg transition-shadow duration-200">

                                            <h3 className="text-xl font-semibold text-blue-700 mb-2">

                                                {plan.destination || 'Unnamed Destination'}

                                                <span className="text-sm font-normal text-gray-500 ml-2">

                                                    ({plan.timestamp?.toDate().toLocaleDateString()})

                                                </span>

                                            </h3>

                                            <p className="text-gray-600 text-sm mb-2">

                                                <span className="font-medium">Interests:</span> {plan.interests || 'N/A'} |

                                                <span className="font-medium ml-2">Budget:</span> {plan.budget || 'N/A'} |

                                                <span className="font-medium ml-2">Season:</span> {plan.season || 'N/A'}

                                            </p>

                                            <div className="flex gap-3 mt-4">

                                                <button

                                                    onClick={() => loadSavedPlan(plan)}

                                                    className="px-4 py-2 bg-indigo-500 text-white rounded-lg hover:bg-indigo-600 transition-colors duration-200 text-sm"

                                                >

                                                    Load Plan

                                                </button>

                                                <button

                                                    onClick={() => deleteSavedPlan(plan.id)}

                                                    className="px-4 py-2 bg-red-500 text-white rounded-lg hover:bg-red-600 transition-colors duration-200 text-sm"

                                                >

                                                    Delete Plan

                                                </button>

                                            </div>

                                        </div>

                                    ))}

                                </div>

                            </div>

                        )}

                    </div>

                </div>

            );

        }

        ReactDOM.render(<App />, document.getElementById('root'));

    </script>

</body>

</html>