Step-by-Step Development Guide: Building an AI-Powered ESG Management Platform

This guide outlines a phased approach to developing a comprehensive AI-powered ESG management platform, leveraging Next.js for the frontend, Firebase (Firestore and Authentication) for backend services and data storage, and Google's Gemini API and Genkit for advanced AI analysis and insights. The development is structured to build a foundational application first, then progressively add sophisticated features, ensuring a complete and robust solution.

Project Setup and Prerequisites

Before diving into development, ensure you have the following set up:

- **Node.js & npm/yarn:** Installed on your machine.
- **Google Cloud Project:** Create a new project in the Google Cloud Console.
- **Firebase Project:** Create a Firebase project within your Google Cloud project.
    - Enable **Firestore Database** (Native mode).
    - Enable **Authentication** (Email/Password, Google Sign-in, etc.).
- **Gemini API Key:** Enable the Gemini API in your Google Cloud project and generate an API key.
- **Genkit CLI:** Install the Genkit CLI globally (npm install -g genkit).

Phase 1: Core Data Ingestion & Storage (Minimum Viable Product - MVP)

This phase focuses on establishing the fundamental structure of your Next.js application, setting up Firebase for data storage (Firestore) and user authentication, and implementing a basic mechanism for manual ESG data input. This forms the backbone of your platform.

Objective

To create a functional Next.js application with secure user authentication and a basic data entry form to manually capture core ESG metrics, storing them in Firestore.

Key Features

- Next.js project initialization.
- Firebase project setup and integration with Next.js.
- User registration and login (Email/Password).
- Firestore database initialization.
- Basic form for manual ESG data entry (e.g., Energy Consumption, Water Usage,

Employee Count).
- Display of submitted data.

Technology Focus

- **Next.js:** Frontend framework.
- **Firebase Authentication:** User management.
- **Firebase Firestore:** NoSQL database for storing ESG data.
- **Tailwind CSS:** For basic styling.

Step-by-Step Implementation

Step 1.1: Initialize Next.js Project

Create a new Next.js project with TypeScript and Tailwind CSS.
```
npx create-next-app esg-platform --typescript --tailwind --eslint
cd esg-platform
```

Step 1.2: Install Firebase and Setup Configuration

Install Firebase SDK and configure it in your Next.js app.
```
npm install firebase
```

Create a firebaseConfig.js file (e.g., in utils/firebaseConfig.js):
```
// utils/firebaseConfig.js
import { initializeApp } from 'firebase/app';
import { getAuth } from 'firebase/auth';
import { getFirestore } from 'firebase/firestore';

// Your web app's Firebase 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"
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
const auth = getAuth(app);
const db = getFirestore(app);

export { app, auth, db };
```

Important: Replace placeholders with your actual Firebase project configuration.

Step 1.3: Implement User Authentication

Create pages for login.js and register.js and integrate Firebase Authentication.

- **pages/register.js:**

```
// pages/register.js
import { useState } from 'react';
import { createUserWithEmailAndPassword } from 'firebase/auth';
import { auth } from '../utils/firebaseConfig';
import Link from 'next/link';

export default function Register() {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState(null);

  const handleRegister = async (e) => {
    e.preventDefault();
    setError(null);
    try {
      await createUserWithEmailAndPassword(auth, email, password);
      alert('Registration successful! You can now log in.');
    } catch (err) {
      setError(err.message);
    }
  };

  return (
    <div className="min-h-screen flex items-center justify-center bg-gray-100">
      <div className="bg-white p-8 rounded-lg shadow-md w-full max-w-md">
        <h2 className="text-2xl font-bold mb-6 text-center">Register</h2>
        <form onSubmit={handleRegister}>
          <div className="mb-4">
            <label className="block text-gray-700 text-sm font-bold mb-2"
htmlFor="email">
              Email
            </label>
            <input
              type="email"
              id="email"
              className="shadow appearance-none border rounded w-full py-2 px-3
text-gray-700 leading-tight focus:outline-none focus:shadow-outline"
```

```
            value={email}
            onChange={(e) => setEmail(e.target.value)}
            required
           />
          </div>
          <div className="mb-6">
           <label className="block text-gray-700 text-sm font-bold mb-2"
htmlFor="password">
            Password
           </label>
           <input
            type="password"
            id="password"
            className="shadow appearance-none border rounded w-full py-2 px-3
text-gray-700 mb-3 leading-tight focus:outline-none focus:shadow-outline"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
            required
           />
          </div>
          {error && <p className="text-red-500 text-xs italic mb-4">{error}</p>}
          <div className="flex items-center justify-between">
           <button
            type="submit"
            className="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4
rounded focus:outline-none focus:shadow-outline"
           >
            Register
           </button>
           <Link href="/login" className="inline-block align-baseline font-bold text-sm
text-blue-500 hover:text-blue-800">
            Already have an account? Login
           </Link>
          </div>
         </form>
        </div>
       </div>
      );
}
```

- **pages/login.js:**

```
// pages/login.js
import { useState } from 'react';
```

```
import { signInWithEmailAndPassword } from 'firebase/auth';
import { auth } from '../utils/firebaseConfig';
import { useRouter } from 'next/router';
import Link from 'next/link';

export default function Login() {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState(null);
  const router = useRouter();

  const handleLogin = async (e) => {
    e.preventDefault();
    setError(null);
    try {
      await signInWithEmailAndPassword(auth, email, password);
      router.push('/'); // Redirect to home page on successful login
    } catch (err) {
      setError(err.message);
    }
  };

  return (
    <div className="min-h-screen flex items-center justify-center bg-gray-100">
      <div className="bg-white p-8 rounded-lg shadow-md w-full max-w-md">
        <h2 className="text-2xl font-bold mb-6 text-center">Login</h2>
        <form onSubmit={handleLogin}>
          <div className="mb-4">
            <label className="block text-gray-700 text-sm font-bold mb-2"
htmlFor="email">
              Email
            </label>
            <input
              type="email"
              id="email"
              className="shadow appearance-none border rounded w-full py-2 px-3
text-gray-700 leading-tight focus:outline-none focus:shadow-outline"
              value={email}
              onChange={(e) => setEmail(e.target.value)}
              required
            />
          </div>
          <div className="mb-6">
```

```
          <label className="block text-gray-700 text-sm font-bold mb-2"
htmlFor="password">
            Password
          </label>
          <input
            type="password"
            id="password"
            className="shadow appearance-none border rounded w-full py-2 px-3
text-gray-700 mb-3 leading-tight focus:outline-none focus:shadow-outline"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
            required
          />
        </div>
        {error && <p className="text-red-500 text-xs italic mb-4">{error}</p>}
        <div className="flex items-center justify-between">
          <button
            type="submit"
            className="bg-green-500 hover:bg-green-700 text-white font-bold py-2 px-4
rounded focus:outline-none focus:shadow-outline"
          >
            Login
          </button>
          <Link href="/register" className="inline-block align-baseline font-bold text-sm
text-blue-500 hover:text-blue-800">
            Don't have an account? Register
          </Link>
        </div>
      </form>
    </div>
  </div>
 );
}
```

Step 1.4: Create Data Entry Form and Display

Modify pages/index.js to include a data entry form and display submitted data. Implement a basic authentication check to redirect unauthenticated users.

```
// pages/index.js
import { useState, useEffect } from 'react';
import { auth, db } from '../utils/firebaseConfig';
import { collection, addDoc, getDocs, query, where, orderBy } from 'firebase/firestore';
import { onAuthStateChanged, signOut } from 'firebase/auth';
```

```
import { useRouter } from 'next/router';

export default function Home() {
  const [user, setUser] = useState(null);
  const [esgData, setEsgData] = useState([]);
  const [energyConsumption, setEnergyConsumption] = useState('');
  const [waterUsage, setWaterUsage] = useState('');
  const [wasteGenerated, setWasteGenerated] = useState('');
  const [submissionError, setSubmissionError] = useState(null);
  const router = useRouter();

  useEffect(() => {
    const unsubscribe = onAuthStateChanged(auth, (currentUser) => {
      if (currentUser) {
        setUser(currentUser);
        fetchEsgData(currentUser.uid);
      } else {
        router.push('/login');
      }
    });
    return () => unsubscribe();
  }, [router]);

  const fetchEsgData = async (userId) => {
    try {
      const q = query(collection(db, 'esg_data'), where('userId', '==', userId),
orderBy('timestamp', 'desc'));
      const querySnapshot = await getDocs(q);
      const data = querySnapshot.docs.map(doc => ({ id: doc.id, ...doc.data() }));
      setEsgData(data);
    } catch (error) {
      console.error("Error fetching ESG data: ", error);
    }
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    setSubmissionError(null);
    if (!user) {
      setSubmissionError("You must be logged in to submit data.");
      return;
    }
```

```
  try {
    await addDoc(collection(db, 'esg_data'), {
      userId: user.uid,
      energyConsumption: parseFloat(energyConsumption),
      waterUsage: parseFloat(waterUsage),
      wasteGenerated: parseFloat(wasteGenerated),
      timestamp: new Date(),
    });
    setEnergyConsumption('');
    setWaterUsage('');
    setWasteGenerated('');
    fetchEsgData(user.uid); // Refresh data after submission
  } catch (err) {
    setSubmissionError(err.message);
    console.error("Error adding document: ", err);
  }
};

const handleLogout = async () => {
  try {
    await signOut(auth);
    router.push('/login');
  } catch (error) {
    console.error("Error logging out: ", error);
  }
};

if (!user) {
  return (
    <div className="min-h-screen flex items-center justify-center bg-gray-100">
      <p>Loading user session...</p>
    </div>
  );
}

return (
  <div className="min-h-screen bg-gray-100 p-8">
    <div className="max-w-4xl mx-auto bg-white p-8 rounded-lg shadow-md">
      <div className="flex justify-between items-center mb-6">
        <h1 className="text-3xl font-bold">ESG Data Platform</h1>
        <button
          onClick={handleLogout}
          className="bg-red-500 hover:bg-red-700 text-white font-bold py-2 px-4 rounded
```

```
focus:outline-none focus:shadow-outline"
        >
          Logout
        </button>
      </div>

      <h2 className="text-xl font-semibold mb-4">Welcome, {user.email}</h2>

      <form onSubmit={handleSubmit} className="mb-8 p-6 border rounded-lg bg-gray-50">
        <h3 className="text-lg font-medium mb-4">Submit New ESG Data</h3>
        <div className="grid grid-cols-1 md:grid-cols-3 gap-4 mb-4">
          <div>
            <label className="block text-gray-700 text-sm font-bold mb-2" htmlFor="energy">
              Energy Consumption (kWh)
            </label>
            <input
              type="number"
              id="energy"
              className="shadow appearance-none border rounded w-full py-2 px-3
text-gray-700 leading-tight focus:outline-none focus:shadow-outline"
              value={energyConsumption}
              onChange={(e) => setEnergyConsumption(e.target.value)}
              required
            />
          </div>
          <div>
            <label className="block text-gray-700 text-sm font-bold mb-2" htmlFor="water">
              Water Usage (liters)
            </label>
            <input
              type="number"
              id="water"
              className="shadow appearance-none border rounded w-full py-2 px-3
text-gray-700 leading-tight focus:outline-none focus:shadow-outline"
              value={waterUsage}
              onChange={(e) => setWaterUsage(e.target.value)}
              required
            />
          </div>
          <div>
            <label className="block text-gray-700 text-sm font-bold mb-2" htmlFor="waste">
              Waste Generated (kg)
            </label>
```

```
            <input
              type="number"
              id="waste"
              className="shadow appearance-none border rounded w-full py-2 px-3
text-gray-700 leading-tight focus:outline-none focus:shadow-outline"
              value={wasteGenerated}
              onChange={(e) => setWasteGenerated(e.target.value)}
              required
            />
          </div>
        </div>
        {submissionError && <p className="text-red-500 text-xs italic
mb-4">{submissionError}</p>}
        <button
          type="submit"
          className="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded
focus:outline-none focus:shadow-outline"
        >
          Submit ESG Data
        </button>
      </form>

      <h3 className="text-lg font-medium mb-4">Your Submitted ESG Data</h3>
      {esgData.length === 0 ? (
        <p>No ESG data submitted yet. Start by adding some above!</p>
      ) : (
        <div className="overflow-x-auto">
          <table className="min-w-full bg-white border border-gray-200 rounded-lg">
            <thead>
              <tr className="bg-gray-100 text-left text-gray-600 uppercase text-sm
leading-normal">
                <th className="py-3 px-6 border-b">Date</th>
                <th className="py-3 px-6 border-b">Energy (kWh)</th>
                <th className="py-3 px-6 border-b">Water (liters)</th>
                <th className="py-3 px-6 border-b">Waste (kg)</th>
              </tr>
            </thead>
            <tbody className="text-gray-700 text-sm">
              {esgData.map((data) => (
                <tr key={data.id} className="border-b border-gray-200 hover:bg-gray-50">
                  <td className="py-3 px-6">{new Date(data.timestamp.seconds *
1000).toLocaleDateString()}</td>
                  <td className="py-3 px-6">{data.energyConsumption}</td>
```

```
                <td className="py-3 px-6">{data.waterUsage}</td>
                <td className="py-3 px-6">{data.wasteGenerated}</td>
              </tr>
            ))}
          </tbody>
        </table>
      </div>
    )}
  </div>
 </div>
 );
}
```

Expected Outcome (Phase 1)

You will have a basic Next.js application where users can register, log in, manually submit simple ESG data (energy, water, waste), and view their submitted data in a table. This establishes the core data flow and user authentication.

Phase 2: Basic Dashboard & User Management

This phase enhances the user experience by introducing a dashboard for visualizing the collected ESG data and implements a more robust user management system with Role-Based Access Control (RBAC).

Objective

To provide visual insights into ESG data through interactive charts and to manage user roles and permissions effectively.

Key Features

- Interactive charts (e.g., line charts for trends, bar charts for comparisons) for ESG metrics.
- Implementation of Role-Based Access Control (RBAC) (e.g., Admin, Data Entry, Viewer).
- Admin interface for managing user roles.
- Conditional rendering of UI elements based on user roles.

Technology Focus

- **Recharts:** For creating interactive data visualizations.
- **Firebase Firestore:** To store user roles.
- **Next.js:** For UI and routing.

Step-by-Step Implementation

Step 2.1: Install Recharts

npm install recharts

Step 2.2: Implement RBAC in Firestore

When a user registers, store their default role (e.g., 'viewer') in a users collection in Firestore. Create an admin function to update roles.

- **Update pages/register.js (after successful user creation):**

```
// In handleRegister function in pages/register.js
import { doc, setDoc } from 'firebase/firestore';
import { db } from '../utils/firebaseConfig'; // Import db

// ...
const handleRegister = async (e) => {
  e.preventDefault();
  setError(null);
  try {
    const userCredential = await createUserWithEmailAndPassword(auth, email,
password);
    // Add user role to Firestore
    await setDoc(doc(db, 'users', userCredential.user.uid), {
      email: userCredential.user.email,
      role: 'data_entry', // Default role
      createdAt: new Date(),
    });
    alert('Registration successful! You can now log in.');
  } catch (err) {
    setError(err.message);
  }
};
// ...
```

- **Create a custom hook or context for user and role management (e.g., hooks/useAuth.js):**

```
// hooks/useAuth.js
import { useState, useEffect } from 'react';
import { onAuthStateChanged } from 'firebase/auth';
import { doc, getDoc } from 'firebase/firestore';
import { auth, db } from '../utils/firebaseConfig';

export const useAuth = () => {
  const [user, setUser] = useState(null);
```

```
  const [role, setRole] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const unsubscribe = onAuthStateChanged(auth, async (currentUser) => {
      if (currentUser) {
        setUser(currentUser);
        try {
          const userDocRef = doc(db, 'users', currentUser.uid);
          const userDocSnap = await getDoc(userDocRef);
          if (userDocSnap.exists()) {
            setRole(userDocSnap.data().role);
          } else {
            console.warn("User document not found for:", currentUser.uid);
            setRole('viewer'); // Default to viewer if doc missing
          }
        } catch (error) {
          console.error("Error fetching user role:", error);
          setRole('viewer'); // Fallback in case of error
        }
      } else {
        setUser(null);
        setRole(null);
      }
      setLoading(false);
    });
    return () => unsubscribe();
  }, []);

  return { user, role, loading };
};
```

Step 2.3: Integrate Dashboard Visualizations

Modify pages/index.js to use useAuth and display charts.

```
// pages/index.js (updated to include charts and useAuth)
import { useState, useEffect } from 'react';
import { auth, db } from '../utils/firebaseConfig';
import { collection, addDoc, getDocs, query, where, orderBy } from 'firebase/firestore';
import { signOut } from 'firebase/auth';
import { useRouter } from 'next/router';
import { LineChart, Line, XAxis, YAxis, CartesianGrid, Tooltip, Legend, ResponsiveContainer }
from 'recharts';
```

```
import { useAuth } from '../hooks/useAuth'; // Import the custom hook

export default function Home() {
  const { user, role, loading } = useAuth(); // Use the custom hook
  const [esgData, setEsgData] = useState([]);
  const [energyConsumption, setEnergyConsumption] = useState('');
  const [waterUsage, setWaterUsage] = useState('');
  const [wasteGenerated, setWasteGenerated] = useState('');
  const [submissionError, setSubmissionError] = useState(null);
  const router = useRouter();

  useEffect(() => {
    if (!loading && !user) {
      router.push('/login');
    } else if (user) {
      fetchEsgData(user.uid);
    }
  }, [user, loading, router]);

  const fetchEsgData = async (userId) => {
    try {
      const q = query(collection(db, 'esg_data'), where('userId', '==', userId),
orderBy('timestamp', 'asc')); // Order by asc for charts
      const querySnapshot = await getDocs(q);
      const data = querySnapshot.docs.map(doc => ({
        id: doc.id,
        ...doc.data(),
        date: new Date(doc.data().timestamp.seconds * 1000).toLocaleDateString(), // Format
date for charts
      }));
      setEsgData(data);
    } catch (error) {
      console.error("Error fetching ESG data: ", error);
    }
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    setSubmissionError(null);
    if (!user) {
      setSubmissionError("You must be logged in to submit data.");
      return;
    }
```

```
  if (role !== 'admin' && role !== 'data_entry') {
    setSubmissionError("You do not have permission to submit data.");
    return;
  }

  try {
    await addDoc(collection(db, 'esg_data'), {
      userId: user.uid,
      energyConsumption: parseFloat(energyConsumption),
      waterUsage: parseFloat(waterUsage),
      wasteGenerated: parseFloat(wasteGenerated),
      timestamp: new Date(),
    });
    setEnergyConsumption('');
    setWaterUsage('');
    setWasteGenerated('');
    fetchEsgData(user.uid); // Refresh data after submission
  } catch (err) {
    setSubmissionError(err.message);
    console.error("Error adding document: ", err);
  }
};

const handleLogout = async () => {
  try {
    await signOut(auth);
    router.push('/login');
  } catch (error) {
    console.error("Error logging out: ", error);
  }
};

if (loading) {
  return (
    <div className="min-h-screen flex items-center justify-center bg-gray-100">
      <p>Loading user session...</p>
    </div>
  );
}

if (!user) {
  return null; // Redirect handled by useEffect
}
```

```
  return (
    <div className="min-h-screen bg-gray-100 p-8">
      <div className="max-w-4xl mx-auto bg-white p-8 rounded-lg shadow-md">
        <div className="flex justify-between items-center mb-6">
          <h1 className="text-3xl font-bold">ESG Data Platform</h1>
          <button
            onClick={handleLogout}
            className="bg-red-500 hover:bg-red-700 text-white font-bold py-2 px-4 rounded
focus:outline-none focus:shadow-outline"
          >
            Logout
          </button>
        </div>

        <h2 className="text-xl font-semibold mb-4">Welcome, {user.email} (Role: {role})</h2>

        {(role === 'admin' || role === 'data_entry') && (
          <form onSubmit={handleSubmit} className="mb-8 p-6 border rounded-lg
bg-gray-50">
            <h3 className="text-lg font-medium mb-4">Submit New ESG Data</h3>
            <div className="grid grid-cols-1 md:grid-cols-3 gap-4 mb-4">
              <div>
                <label className="block text-gray-700 text-sm font-bold mb-2" htmlFor="energy">
                  Energy Consumption (kWh)
                </label>
                <input
                  type="number"
                  id="energy"
                  className="shadow appearance-none border rounded w-full py-2 px-3
text-gray-700 leading-tight focus:outline-none focus:shadow-outline"
                  value={energyConsumption}
                  onChange={(e) => setEnergyConsumption(e.target.value)}
                  required
                />
              </div>
              <div>
                <label className="block text-gray-700 text-sm font-bold mb-2" htmlFor="water">
                  Water Usage (liters)
                </label>
                <input
                  type="number"
                  id="water"
```

```
              className="shadow appearance-none border rounded w-full py-2 px-3
text-gray-700 leading-tight focus:outline-none focus:shadow-outline"
              value={waterUsage}
              onChange={(e) => setWaterUsage(e.target.value)}
              required
            />
          </div>
          <div>
            <label className="block text-gray-700 text-sm font-bold mb-2" htmlFor="waste">
              Waste Generated (kg)
            </label>
            <input
              type="number"
              id="waste"
              className="shadow appearance-none border rounded w-full py-2 px-3
text-gray-700 leading-tight focus:outline-none focus:shadow-outline"
              value={wasteGenerated}
              onChange={(e) => setWasteGenerated(e.target.value)}
              required
            />
          </div>
        </div>
        {submissionError && <p className="text-red-500 text-xs italic
mb-4">{submissionError}</p>}
        <button
          type="submit"
          className="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded
focus:outline-none focus:shadow-outline"
        >
          Submit ESG Data
        </button>
      </form>
    )}

    <h3 className="text-lg font-medium mb-4">ESG Data Trends</h3>
    {esgData.length === 0 ? (
      <p>No data available to display charts. Submit some ESG data first.</p>
    ) : (
      <div className="grid grid-cols-1 lg:grid-cols-2 gap-8 mb-8">
        <div className="bg-gray-50 p-4 rounded-lg shadow-sm">
          <h4 className="text-md font-semibold mb-2">Energy Consumption Over Time</h4>
          <ResponsiveContainer width="100%" height={300}>
            <LineChart data={esgData}>
```

```
              <CartesianGrid strokeDasharray="3 3" />
              <XAxis dataKey="date" />
              <YAxis />
              <Tooltip />
              <Legend />
              <Line type="monotone" dataKey="energyConsumption" stroke="#8884d8"
name="Energy (kWh)" />
            </LineChart>
          </ResponsiveContainer>
        </div>
        <div className="bg-gray-50 p-4 rounded-lg shadow-sm">
          <h4 className="text-md font-semibold mb-2">Water Usage Over Time</h4>
          <ResponsiveContainer width="100%" height={300}>
            <LineChart data={esgData}>
              <CartesianGrid strokeDasharray="3 3" />
              <XAxis dataKey="date" />
              <YAxis />
              <Tooltip />
              <Legend />
              <Line type="monotone" dataKey="waterUsage" stroke="#82ca9d" name="Water
(liters)" />
            </LineChart>
          </ResponsiveContainer>
        </div>
        <div className="bg-gray-50 p-4 rounded-lg shadow-sm lg:col-span-2">
          <h4 className="text-md font-semibold mb-2">Waste Generated Over Time</h4>
          <ResponsiveContainer width="100%" height={300}>
            <LineChart data={esgData}>
              <CartesianGrid strokeDasharray="3 3" />
              <XAxis dataKey="date" />
              <YAxis />
              <Tooltip />
              <Legend />
              <Line type="monotone" dataKey="wasteGenerated" stroke="#ffc658"
name="Waste (kg)" />
            </LineChart>
          </ResponsiveContainer>
        </div>
      </div>
    )}

    <h3 className="text-lg font-medium mb-4">Your Submitted ESG Data</h3>
    {esgData.length === 0 ? (
```

```
        <p>No ESG data submitted yet. Start by adding some above!</p>
      ) : (
        <div className="overflow-x-auto">
          <table className="min-w-full bg-white border border-gray-200 rounded-lg">
            <thead>
              <tr className="bg-gray-100 text-left text-gray-600 uppercase text-sm
leading-normal">
                <th className="py-3 px-6 border-b">Date</th>
                <th className="py-3 px-6 border-b">Energy (kWh)</th>
                <th className="py-3 px-6 border-b">Water (liters)</th>
                <th className="py-3 px-6 border-b">Waste (kg)</th>
              </tr>
            </thead>
            <tbody className="text-gray-700 text-sm">
              {esgData.map((data) => (
                <tr key={data.id} className="border-b border-gray-200 hover:bg-gray-50">
                  <td className="py-3 px-6">{data.date}</td>
                  <td className="py-3 px-6">{data.energyConsumption}</td>
                  <td className="py-3 px-6">{data.waterUsage}</td>
                  <td className="py-3 px-6">{data.wasteGenerated}</td>
                </tr>
              ))}
            </tbody>
          </table>
        </div>
      )}

      {role === 'admin' && (
        <div className="mt-8 p-6 border rounded-lg bg-red-50">
          <h3 className="text-lg font-medium mb-4">Admin: Manage User Roles</h3>
          {/* This is a placeholder. In a real app, you'd fetch all users and allow role editing. */}
          <p className="text-gray-700">
            Admin functionality to manage user roles would go here. You would fetch all users
from the `users` collection and provide an interface to update their `role` field.
          </p>
          <p className="text-sm text-gray-600 mt-2">
            Example: Fetch all users, display them in a table, and add a dropdown or button to
change their role (e.g., to 'admin', 'data_entry', 'viewer').
          </p>
        </div>
      )}
    </div>
  </div>
```

```
  );
}
```

Expected Outcome (Phase 2)

Your application will now display interactive charts of ESG data trends. Users will have assigned roles, and the data submission form will only be visible to 'admin' and 'data_entry' roles. An admin section placeholder will be present, indicating where role management functionality would be built.

Phase 3: AI-Powered Data Cleaning & Initial Analysis

This phase integrates the Gemini API to enhance data quality through anomaly detection and provide initial AI-driven insights from the collected ESG data.

Objective

To leverage AI for automated data validation, cleaning, and basic analytical insights, improving data reliability and providing preliminary intelligence.

Key Features

- Backend API route in Next.js to interact with Gemini API.
- Functionality to send ESG data to Gemini for anomaly detection.
- Gemini API for basic text analysis (e.g., identifying qualitative insights from a simple text input field).
- Display of AI-generated data quality flags and insights on the dashboard.

Technology Focus

- **Gemini API:** For AI-powered text analysis and data validation.
- **Next.js API Routes:** To handle server-side interactions with Gemini.
- **Firebase Firestore:** To store AI analysis results alongside ESG data.

Step-by-Step Implementation

Step 3.1: Configure Gemini Genkit (Optional but Recommended for Complex Flows)

While you can call Gemini API directly, Genkit helps manage AI workflows. For this phase, we'll show a direct API call, but keep Genkit in mind for Phase 4.

Step 3.2: Create a Next.js API Route for Gemini Interaction

Create pages/api/analyze-esg.js to handle requests to Gemini.

```
// pages/api/analyze-esg.js
import { GoogleGenerativeAI } from '@google/generative-ai';
```

```
const genAI = new GoogleGenerativeAI(process.env.GEMINI_API_KEY);
const model = genAI.getGenerativeModel({ model: "gemini-pro" }); // Use gemini-pro for text tasks

export default async function handler(req, res) {
  if (req.method === 'POST') {
    const { energyConsumption, waterUsage, wasteGenerated } = req.body;

    if (energyConsumption === undefined || waterUsage === undefined || wasteGenerated === undefined) {
      return res.status(400).json({ error: 'Missing ESG data parameters' });
    }

    try {
      // Prompt for anomaly detection and basic insight
      const prompt = `Analyze the following ESG data for potential anomalies and provide a brief insight:\n
        Energy Consumption: ${energyConsumption} kWh\n
        Water Usage: ${waterUsage} liters\n
        Waste Generated: ${wasteGenerated} kg\n
        Consider typical corporate operations. Is any value unusually high or low? What's a quick insight based on these numbers?
        Respond in a JSON format like: {"anomaly_flag": true/false, "anomaly_details": "...", "insight": "..."}`;

      const result = await model.generateContent(prompt);
      const response = await result.response;
      const text = response.text();

      // Attempt to parse JSON. Gemini might return markdown JSON or plain text.
      let parsedResponse;
      try {
        // Remove markdown backticks if present
        const cleanText = text.replace(/```json\n|\n```/g, '');
        parsedResponse = JSON.parse(cleanText);
      } catch (parseError) {
        console.error("Failed to parse Gemini response as JSON, using raw text:", parseError);
        parsedResponse = {
          anomaly_flag: true, // Default to true if parsing fails, requires human review
          anomaly_details: "AI response parsing error or unexpected format. Raw response: " + text,
          insight: "AI analysis could not be fully processed due to formatting issues. Review raw response."
```

```
      };
    }

    res.status(200).json(parsedResponse);
  } catch (error) {
    console.error('Error calling Gemini API:', error);
    res.status(500).json({ error: 'Failed to analyze ESG data with AI', details: error.message });
  }
 } else {
  res.setHeader('Allow', ['POST']);
  res.status(405).end(`Method ${req.method} Not Allowed`);
 }
}
```

Important: Add GEMINI_API_KEY=your_gemini_api_key to your .env.local file.

Step 3.3: Integrate AI Analysis into Data Submission

Modify pages/index.js to call the new API route after data submission and display the results.

```
// pages/index.js (updated to include AI analysis)
// ... (existing imports and useAuth hook)
import { useState, useEffect } from 'react';
import { auth, db } from '../utils/firebaseConfig';
import { collection, addDoc, getDocs, query, where, orderBy, updateDoc, doc } from
'firebase/firestore'; // Added updateDoc, doc
import { signOut } from 'firebase/auth';
import { useRouter } from 'next/router';
import { LineChart, Line, XAxis, YAxis, CartesianGrid, Tooltip, Legend, ResponsiveContainer }
from 'recharts';
import { useAuth } from '../hooks/useAuth';

export default function Home() {
 const { user, role, loading } = useAuth();
 const [esgData, setEsgData] = useState([]);
 const [energyConsumption, setEnergyConsumption] = useState('');
 const [waterUsage, setWaterUsage] = useState('');
 const [wasteGenerated, setWasteGenerated] = useState('');
 const [submissionError, setSubmissionError] = useState(null);
 const [aiAnalysisResult, setAiAnalysisResult] = useState(null); // New state for AI analysis
 const [isAnalyzing, setIsAnalyzing] = useState(false); // New state for loading indicator
 const router = useRouter();

 useEffect(() => {
  if (!loading && !user) {
```

```
      router.push('/login');
    } else if (user) {
      fetchEsgData(user.uid);
    }
  }, [user, loading, router]);

  const fetchEsgData = async (userId) => {
    try {
      const q = query(collection(db, 'esg_data'), where('userId', '==', userId),
orderBy('timestamp', 'asc'));
      const querySnapshot = await getDocs(q);
      const data = querySnapshot.docs.map(doc => ({
        id: doc.id,
        ...doc.data(),
        date: new Date(doc.data().timestamp.seconds * 1000).toLocaleDateString(),
      }));
      setEsgData(data);
    } catch (error) {
      console.error("Error fetching ESG data: ", error);
    }
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    setSubmissionError(null);
    setAiAnalysisResult(null); // Clear previous analysis
    setIsAnalyzing(true);

    if (!user) {
      setSubmissionError("You must be logged in to submit data.");
      setIsAnalyzing(false);
      return;
    }
    if (role !== 'admin' && role !== 'data_entry') {
      setSubmissionError("You do not have permission to submit data.");
      setIsAnalyzing(false);
      return;
    }

    try {
      // 1. Add data to Firestore
      const newDocRef = await addDoc(collection(db, 'esg_data'), {
        userId: user.uid,
```

```
    energyConsumption: parseFloat(energyConsumption),
    waterUsage: parseFloat(waterUsage),
    wasteGenerated: parseFloat(wasteGenerated),
    timestamp: new Date(),
    aiAnalysis: null, // Placeholder for AI analysis
  });

  // 2. Call AI analysis API
  const response = await fetch('/api/analyze-esg', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      energyConsumption: parseFloat(energyConsumption),
      waterUsage: parseFloat(waterUsage),
      wasteGenerated: parseFloat(wasteGenerated),
    }),
  });

  if (!response.ok) {
    const errorData = await response.json();
    throw new Error(errorData.error || 'Failed to get AI analysis');
  }

  const analysis = await response.json();
  setAiAnalysisResult(analysis);

  // 3. Update the Firestore document with AI analysis
  await updateDoc(doc(db, 'esg_data', newDocRef.id), {
    aiAnalysis: analysis,
  });

  setEnergyConsumption('');
  setWaterUsage('');
  setWasteGenerated('');
  fetchEsgData(user.uid); // Refresh data after submission
} catch (err) {
  setSubmissionError(err.message);
  console.error("Error during data submission or AI analysis: ", err);
} finally {
  setIsAnalyzing(false);
}
```

```
  };

  const handleLogout = async () => {
    try {
      await signOut(auth);
      router.push('/login');
    } catch (error) {
      console.error("Error logging out: ", error);
    }
  };

  if (loading) {
    return (
      <div className="min-h-screen flex items-center justify-center bg-gray-100">
        <p>Loading user session...</p>
      </div>
    );
  }

  if (!user) {
    return null;
  }

  return (
    <div className="min-h-screen bg-gray-100 p-8">
      <div className="max-w-4xl mx-auto bg-white p-8 rounded-lg shadow-md">
        <div className="flex justify-between items-center mb-6">
          <h1 className="text-3xl font-bold">ESG Data Platform</h1>
          <button
            onClick={handleLogout}
            className="bg-red-500 hover:bg-red-700 text-white font-bold py-2 px-4 rounded
focus:outline-none focus:shadow-outline"
          >
            Logout
          </button>
        </div>

        <h2 className="text-xl font-semibold mb-4">Welcome, {user.email} (Role: {role})</h2>

        {(role === 'admin' || role === 'data_entry') && (
          <form onSubmit={handleSubmit} className="mb-8 p-6 border rounded-lg
bg-gray-50">
            <h3 className="text-lg font-medium mb-4">Submit New ESG Data</h3>
```

```
        <div className="grid grid-cols-1 md:grid-cols-3 gap-4 mb-4">
          <div>
            <label className="block text-gray-700 text-sm font-bold mb-2" htmlFor="energy">
              Energy Consumption (kWh)
            </label>
            <input
              type="number"
              id="energy"
              className="shadow appearance-none border rounded w-full py-2 px-3
text-gray-700 leading-tight focus:outline-none focus:shadow-outline"
              value={energyConsumption}
              onChange={(e) => setEnergyConsumption(e.target.value)}
              required
            />
          </div>
          <div>
            <label className="block text-gray-700 text-sm font-bold mb-2" htmlFor="water">
              Water Usage (liters)
            </label>
            <input
              type="number"
              id="water"
              className="shadow appearance-none border rounded w-full py-2 px-3
text-gray-700 leading-tight focus:outline-none focus:shadow-outline"
              value={waterUsage}
              onChange={(e) => setWaterUsage(e.target.value)}
              required
            />
          </div>
          <div>
            <label className="block text-gray-700 text-sm font-bold mb-2" htmlFor="waste">
              Waste Generated (kg)
            </label>
            <input
              type="number"
              id="waste"
              className="shadow appearance-none border rounded w-full py-2 px-3
text-gray-700 leading-tight focus:outline-none focus:shadow-outline"
              value={wasteGenerated}
              onChange={(e) => setWasteGenerated(e.target.value)}
              required
            />
          </div>
```

```
        </div>
        {submissionError && <p className="text-red-500 text-xs italic
mb-4">{submissionError}</p>}
        <button
          type="submit"
          className="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded
focus:outline-none focus:shadow-outline"
          disabled={isAnalyzing}
        >
          {isAnalyzing ? 'Submitting & Analyzing...' : 'Submit ESG Data'}
        </button>
        {aiAnalysisResult && (
          <div className="mt-4 p-4 bg-blue-100 border border-blue-200 rounded-lg">
            <h4 className="font-semibold text-blue-800">AI Analysis Result:</h4>
            <p><strong>Anomaly Detected:</strong> {aiAnalysisResult.anomaly_flag ? 'Yes' :
'No'}</p>
            {aiAnalysisResult.anomaly_details && <p><strong>Details:</strong>
{aiAnalysisResult.anomaly_details}</p>}
            <p><strong>Insight:</strong> {aiAnalysisResult.insight}</p>
          </div>
        )}
      </form>
    )}

    <h3 className="text-lg font-medium mb-4">ESG Data Trends</h3>
    {/* ... (charts code remains the same) ... */}
    {esgData.length === 0 ? (
      <p>No data available to display charts. Submit some ESG data first.</p>
    ) : (
      <div className="grid grid-cols-1 lg:grid-cols-2 gap-8 mb-8">
        <div className="bg-gray-50 p-4 rounded-lg shadow-sm">
          <h4 className="text-md font-semibold mb-2">Energy Consumption Over Time</h4>
          <ResponsiveContainer width="100%" height={300}>
            <LineChart data={esgData}>
              <CartesianGrid strokeDasharray="3 3" />
              <XAxis dataKey="date" />
              <YAxis />
              <Tooltip />
              <Legend />
              <Line type="monotone" dataKey="energyConsumption" stroke="#8884d8"
name="Energy (kWh)" />
            </LineChart>
          </ResponsiveContainer>
```

```
          </div>
          <div className="bg-gray-50 p-4 rounded-lg shadow-sm">
            <h4 className="text-md font-semibold mb-2">Water Usage Over Time</h4>
            <ResponsiveContainer width="100%" height={300}>
              <LineChart data={esgData}>
                <CartesianGrid strokeDasharray="3 3" />
                <XAxis dataKey="date" />
                <YAxis />
                <Tooltip />
                <Legend />
                <Line type="monotone" dataKey="waterUsage" stroke="#82ca9d" name="Water
(liters)" />
              </LineChart>
            </ResponsiveContainer>
          </div>
          <div className="bg-gray-50 p-4 rounded-lg shadow-sm lg:col-span-2">
            <h4 className="text-md font-semibold mb-2">Waste Generated Over Time</h4>
            <ResponsiveContainer width="100%" height={300}>
              <LineChart data={esgData}>
                <CartesianGrid strokeDasharray="3 3" />
                <XAxis dataKey="date" />
                <YAxis />
                <Tooltip />
                <Legend />
                <Line type="monotone" dataKey="wasteGenerated" stroke="#ffc658"
name="Waste (kg)" />
              </LineChart>
            </ResponsiveContainer>
          </div>
        </div>
      )}


      <h3 className="text-lg font-medium mb-4">Your Submitted ESG Data</h3>
      {esgData.length === 0 ? (
        <p>No ESG data submitted yet. Start by adding some above!</p>
      ) : (
        <div className="overflow-x-auto">
          <table className="min-w-full bg-white border border-gray-200 rounded-lg">
            <thead>
              <tr className="bg-gray-100 text-left text-gray-600 uppercase text-sm
leading-normal">
                <th className="py-3 px-6 border-b">Date</th>
```

```
              <th className="py-3 px-6 border-b">Energy (kWh)</th>
              <th className="py-3 px-6 border-b">Water (liters)</th>
              <th className="py-3 px-6 border-b">Waste (kg)</th>
              <th className="py-3 px-6 border-b">AI Anomaly</th>
              <th className="py-3 px-6 border-b">AI Insight</th>
            </tr>
          </thead>
          <tbody className="text-gray-700 text-sm">
            {esgData.map((data) => (
              <tr key={data.id} className="border-b border-gray-200 hover:bg-gray-50">
                <td className="py-3 px-6">{data.date}</td>
                <td className="py-3 px-6">{data.energyConsumption}</td>
                <td className="py-3 px-6">{data.waterUsage}</td>
                <td className="py-3 px-6">{data.wasteGenerated}</td>
                <td className="py-3 px-6">
                  {data.aiAnalysis?.anomaly_flag === true ? (
                    <span className="text-red-600 font-bold">Yes</span>
                  ) : data.aiAnalysis?.anomaly_flag === false ? (
                    <span className="text-green-600">No</span>
                  ) : (
                    'N/A'
                  )}
                </td>
                <td className="py-3 px-6">
                  {data.aiAnalysis?.insight || 'No insight available'}
                </td>
              </tr>
            ))}
          </tbody>
        </table>
      </div>
    )}

    {role === 'admin' && (
      <div className="mt-8 p-6 border rounded-lg bg-red-50">
        <h3 className="text-lg font-medium mb-4">Admin: Manage User Roles</h3>
        <p className="text-gray-700">
          Admin functionality to manage user roles would go here. You would fetch all users from the `users` collection and provide an interface to update their `role` field.
        </p>
        <p className="text-sm text-gray-600 mt-2">
          Example: Fetch all users, display them in a table, and add a dropdown or button to change their role (e.g., to 'admin', 'data_entry', 'viewer').
```

```
          </p>
        </div>
      )}
    </div>
  </div>
 );
}
```

Expected Outcome (Phase 3)

When new ESG data is submitted, it will be sent to a Next.js API route, which then calls the Gemini API for anomaly detection and basic insights. These AI-generated findings will be displayed on the dashboard and stored with the ESG data in Firestore.

Phase 4: Advanced Data Integration & AI Analysis

This phase significantly expands the platform's capabilities by integrating diverse external data sources and leveraging Gemini Genkit for more complex AI workflows, such as processing unstructured documents and performing predictive analytics.

Objective

To automate data gathering from various internal and external sources (including unstructured data) and apply advanced AI models for deeper analysis, risk forecasting, and strategic insights.

Key Features

- Integration with external APIs (e.g., for financial data, third-party ESG ratings).
- File upload functionality for unstructured documents (PDFs, reports).
- AI-powered OCR and NLP for extracting data from unstructured documents using Gemini.
- Predictive analytics for ESG risks (e.g., supply chain compliance, emissions trends) using Gemini Genkit.
- Materiality assessment enhancement using AI.

Technology Focus

- **Google Cloud Storage:** For storing uploaded documents.
- **Next.js API Routes:** For handling file uploads and triggering AI workflows.
- **Gemini Genkit:** For orchestrating multi-step AI pipelines (e.g., OCR -> NLP -> Data Extraction).
- **Firebase Firestore:** To store extracted data and AI analysis results.
- **Third-party APIs/Libraries:** For specific data integrations (conceptual, as specific APIs vary).

Step-by-Step Implementation (Conceptual)

Step 4.1: Setup Google Cloud Storage for Document Uploads

Enable Cloud Storage in your Google Cloud project. You'll need to configure Firebase Storage or use Google Cloud Storage directly.

Step 4.2: Implement File Upload and Processing

Create a new Next.js API route (/api/upload-document) to handle file uploads. This route will:

1. Receive the uploaded document (e.g., PDF).
2. Store it in Google Cloud Storage.
3. Trigger an AI workflow using Gemini Genkit (or a direct Gemini call for OCR/NLP).

- **Conceptual pages/api/upload-document.js:**

```
// pages/api/upload-document.js (Conceptual)
// This requires a file upload library like 'multer' or 'formidable' for Node.js backend
// and Google Cloud Storage client library.
// For simplicity, we'll outline the flow.

import { Storage } from '@google-cloud/storage';
import { GoogleGenerativeAI } from '@google/generative-ai';
// Potentially Genkit imports if you set up a Genkit flow

const storage = new Storage();
const bucketName = 'your-gcs-bucket-name'; // Replace with your bucket
const genAI = new GoogleGenerativeAI(process.env.GEMINI_API_KEY);
const model = genAI.getGenerativeModel({ model: "gemini-pro-vision" }); // For
image/document analysis

export const config = {
  api: {
    bodyParser: false, // Disable default bodyParser for file uploads
  },
};

export default async function handler(req, res) {
  if (req.method === 'POST') {
    // Parse multipart form data to get the file
    // Example using 'formidable' (install with npm install formidable)
    const form = new formidable.IncomingForm();
    form.parse(req, async (err, fields, files) => {
      if (err) {
        console.error("Error parsing form:", err);
        return res.status(500).json({ error: 'Failed to parse form data' });
```

```
    }

    const file = files.esgDocument[0]; // Assuming 'esgDocument' is the field name
    if (!file) {
      return res.status(400).json({ error: 'No file uploaded' });
    }

    const fileName = `${Date.now()}-${file.originalFilename}`;
    const blob = storage.bucket(bucketName).file(fileName);
    const blobStream = blob.createWriteStream();

    blobStream.on('error', (err) => {
      console.error('Error uploading to GCS:', err);
      res.status(500).json({ error: 'Failed to upload document' });
    });

    blobStream.on('finish', async () => {
      const publicUrl = `gs://${bucketName}/${fileName}`; // GCS URI

      try {
        // Use Gemini-Pro-Vision for OCR/NLP on the document
        // For a real PDF, you'd need to convert pages to images or extract text first.
        // For simplicity, let's assume we're sending text content if available, or image data.
        // If it's a PDF, you'd need a library like 'pdf-parse' to extract text or render to
image.

        // Example: If you have base64 image data from a PDF page
        // const imageParts = [
        //   {
        //     inlineData: {
        //       mimeType: "image/jpeg", // or image/png
        //       data: base64ImageData
        //     }
        //   },
        // ];
        // const prompt = "Extract key ESG metrics (e.g., carbon emissions, water usage,
employee diversity) from this document and summarize any sustainability initiatives.
Provide a JSON object with extracted data.";
        // const result = await model.generateContent([prompt, ...imageParts]);

        // For this conceptual example, let's just acknowledge the upload and suggest
next steps
        const aiResponse = {
```

```
            message: `Document uploaded successfully to ${publicUrl}. AI analysis would be
triggered here.`,
            extractedData: { /* AI extracted data will go here */ },
            insights: "AI will analyze this document for ESG data, anomalies, and insights."
          };

          // Store analysis results in Firestore
          // await addDoc(collection(db, 'extracted_esg_data'), {
          //   userId: req.query.userId, // Pass userId from frontend
          //   documentUrl: publicUrl,
          //   aiAnalysis: aiResponse,
          //   timestamp: new Date(),
          // });

          res.status(200).json(aiResponse);
        } catch (aiError) {
          console.error('Error during AI analysis:', aiError);
          res.status(500).json({ error: 'Failed to analyze document with AI', details:
aiError.message });
        }
      });

      // Pipe the file stream to GCS
      fs.createReadStream(file.filepath).pipe(blobStream);
    });
  } else {
    res.setHeader('Allow', ['POST']);
    res.status(405).end(`Method ${req.method} Not Allowed`);
  }
}
```

Note: Handling file uploads in Next.js API routes requires specific libraries (e.g., formidable or multer for Node.js) and careful setup. This is a conceptual outline.

Step 4.3: Implement Advanced AI Analysis with Gemini Genkit

Genkit allows you to define and run AI flows. For example, a flow could:

1. Take a document from Cloud Storage.
2. Use a vision model (Gemini-Pro-Vision) for OCR to extract text.
3. Use a language model (Gemini-Pro) for NLP to extract entities (e.g., company names, emission values, dates), perform sentiment analysis on qualitative data, and identify key sustainability initiatives.
4. Run a separate ML model (trained on historical data) for predictive risk assessment (e.g., likelihood of supply chain disruption based on supplier ESG data).

5. Store the structured, extracted data and predictions back into Firestore.

- **Conceptual Genkit Flow (e.g., in a genkit/flows/esgFlow.ts file):**

```
// genkit/flows/esgFlow.ts (Conceptual)
import { defineFlow, generate, read, write } from '@genkit-ai/core';
import { geminiPro, geminiProVision } from '@genkit-ai/google-cloud';
import { z } from 'zod';

export const esgDocumentAnalysisFlow = defineFlow(
 {
  name: 'esgDocumentAnalysisFlow',
  inputSchema: z.object({
   documentUrl: z.string().describe('URL of the document in Google Cloud Storage'),
   userId: z.string(),
  }),
  outputSchema: z.object({
   extractedData: z.record(z.any()), // Extracted structured ESG data
   risks: z.array(z.string()), // Identified risks
   opportunities: z.array(z.string()), // Identified opportunities
   summary: z.string(), // AI-generated summary
  }),
 },
 async (input) => {
  // Step 1: Read document from GCS (conceptual)
  // In a real scenario, you'd fetch the document content/image from the URL
  // For vision models, you might need to convert PDF pages to images.
  const documentContent = `Content from ${input.documentUrl}`; // Placeholder

  // Step 2: OCR and Initial Text Extraction (using geminiProVision)
  const ocrResult = await generate({
   model: geminiProVision,
   prompt: [
    { text: `Perform OCR and extract all text from this document. Identify any tables and
their content. Document URL: ${input.documentUrl}` },
    // Add image parts if you're sending image data
   ],
   config: { temperature: 0.1 },
  });
  const extractedText = ocrResult.text();

  // Step 3: NLP for Entity Extraction and Sentiment Analysis (using geminiPro)
  const nlpResult = await generate({
   model: geminiPro,
   prompt: `From the following text, extract key ESG metrics (e.g., carbon emissions,
```

```
water usage, employee diversity, supplier names, policy mentions), identify any
sustainability initiatives, and assess overall sentiment regarding environmental and
social performance. Output as JSON. Text: ${extractedText}`,
    config: { temperature: 0.2 },
  });
  const extractedData = JSON.parse(nlpResult.text()); // Assuming JSON output

  // Step 4: Predictive Risk Analysis (conceptual - could be another ML model)
  // This could be a separate Genkit flow or a call to a deployed ML model
  const predictiveRisk = await generate({
    model: geminiPro, // Or a custom model
    prompt: `Based on the extracted data: ${JSON.stringify(extractedData)}, identify
potential ESG risks (e.g., supply chain compliance, regulatory non-compliance,
reputational damage) and opportunities.`,
    config: { temperature: 0.3 },
  });
  const risksAndOpportunities = JSON.parse(predictiveRisk.text());

  // Step 5: Summarize and Return
  const summaryResult = await generate({
    model: geminiPro,
    prompt: `Summarize the key ESG insights, risks, and opportunities from the
document.`,
    config: { temperature: 0.4 },
  });

  return {
    extractedData: extractedData,
    risks: risksAndOpportunities.risks,
    opportunities: risksAndOpportunities.opportunities,
    summary: summaryResult.text(),
  };
 }
);
```

You would then run this flow from your Next.js API route.

Step 4.4: Integrate External Data Providers

For third-party ESG data providers (MSCI, Sustainalytics), you would typically use their provided APIs. Create Next.js API routes that:

1. Call the external API.
2. Receive the data.
3. Normalize and store it in your Firestore database, linking it to relevant

companies/entities.

Expected Outcome (Phase 4)

The platform can now ingest data from various sources, including unstructured documents, and apply advanced AI for comprehensive analysis, risk prediction, and deeper insights, enriching the ESG data profile.

Phase 5: Automated Compliance Reporting & Auditability

This phase focuses on transforming the analyzed ESG data into automated, auditable compliance reports, aligning with various global frameworks.

Objective

To generate compliance reports automatically, tailored to specific ESG frameworks, and ensure full auditability and traceability of all data and changes.

Key Features

- Configuration for various ESG reporting frameworks (GRI, SASB, TCFD, CSRD).
- Automatic generation of report drafts based on collected and analyzed data.
- Generative AI for narrative drafting and pre-filling questionnaires.
- XBRL integration for machine-readable reports (conceptual).
- Robust audit trails for all data modifications and report generations.
- Version control for reports.

Technology Focus

- **Gemini API:** For generative AI (narrative drafting, pre-filling).
- **Next.js:** For UI to configure reports and display generated drafts.
- **PDF Generation Libraries:** (e.g., jspdf, react-pdf/renderer) for creating downloadable reports.
- **Firebase Firestore:** To store report configurations, generated drafts, and audit logs.

Step-by-Step Implementation (Conceptual)

Step 5.1: Define Reporting Frameworks and Data Mapping

In your Firestore, create a reporting_frameworks collection. Each document would define the structure and required data points for a specific framework (e.g., GRI, SASB).

Step 5.2: Implement Report Generation Logic

Create a Next.js API route (/api/generate-report) that:

1. Takes a selected framework and a time period as input.
2. Fetches relevant ESG data from Firestore.

3. Uses Gemini API for narrative generation for qualitative sections.
    - **Prompt Example for Narrative:** "Based on the following ESG data for Q1 2024: Energy Consumption: 1500 kWh, Water Usage: 500 liters, Waste Generated: 100 kg, and AI insight: 'Energy consumption is slightly higher than average due to increased production.', draft a concise summary of environmental performance for a sustainability report, focusing on key metrics and identified areas for improvement."
4. Populates a report template (using a PDF generation library).
5. Stores the generated report (e.g., as a PDF in Cloud Storage) and a record of its generation in Firestore (for auditability).

- **Conceptual pages/api/generate-report.js:**

```
// pages/api/generate-report.js (Conceptual)
import { GoogleGenerativeAI } from '@google/generative-ai';
import { getDocs, collection, query, where } from 'firebase/firestore';
import { db } from '../../utils/firebaseConfig';
// import jsPDF from 'jspdf'; // For server-side PDF generation (or use client-side)

const genAI = new GoogleGenerativeAI(process.env.GEMINI_API_KEY);
const model = genAI.getGenerativeModel({ model: "gemini-pro" });

export default async function handler(req, res) {
 if (req.method === 'POST') {
  const { userId, framework, startDate, endDate } = req.body;

  if (!userId || !framework || !startDate || !endDate) {
   return res.status(400).json({ error: 'Missing parameters for report generation' });
  }

  try {
   // 1. Fetch relevant ESG data for the period
   const q = query(
    collection(db, 'esg_data'),
    where('userId', '==', userId),
    where('timestamp', '>=', new Date(startDate)),
    where('timestamp', '<=', new Date(endDate))
   );
   const querySnapshot = await getDocs(q);
   const rawEsgData = querySnapshot.docs.map(doc => doc.data());

   // 2. Aggregate and prepare data for the chosen framework
   // This logic would be complex, mapping rawEsgData to framework requirements
   const aggregatedData = {
    totalEnergy: rawEsgData.reduce((sum, d) => sum + (d.energyConsumption || 0), 0),
```

```
      totalWater: rawEsgData.reduce((sum, d) => sum + (d.waterUsage || 0), 0),
      totalWaste: rawEsgData.reduce((sum, d) => sum + (d.wasteGenerated || 0), 0),
      // ... more metrics
    };

    // 3. Use Generative AI for narrative sections
    const narrativePrompt = `Based on the following aggregated ESG data for the period ${startDate} to ${endDate} and the ${framework} framework: ${JSON.stringify(aggregatedData)}. Draft a concise summary of the environmental performance and any key social/governance highlights. Focus on trends and areas for improvement.`;
    const result = await model.generateContent(narrativePrompt);
    const response = await result.response;
    const aiNarrative = response.text();

    // 4. Generate the report (e.g., PDF)
    // This is highly conceptual and depends on your PDF library.
    // const doc = new jsPDF();
    // doc.text("ESG Report - " + framework, 10, 10);
    // doc.text("Period: " + startDate + " to " + endDate, 10, 20);
    // doc.text("Environmental Summary:", 10, 40);
    // doc.text(aiNarrative, 10, 50);
    // const pdfBlob = doc.output('blob'); // Or base64 string

    // 5. Store audit trail and potentially the generated report URL
    // await addDoc(collection(db, 'report_audits'), {
    //   userId: userId,
    //   framework: framework,
    //   period: { startDate, endDate },
    //   generatedAt: new Date(),
    //   aiNarrativeUsed: aiNarrative,
    //   // reportUrl: 'URL_TO_GENERATED_PDF_IN_GCS',
    //   // ... other audit details
    // });

    res.status(200).json({
      message: `Report draft for ${framework} generated successfully.`,
      aiNarrative: aiNarrative,
      // reportDownloadUrl: 'URL_TO_GENERATED_PDF_IN_GCS',
    });
  } catch (error) {
    console.error('Error generating report:', error);
    res.status(500).json({ error: 'Failed to generate report', details: error.message });
```

```
    }
  } else {
    res.setHeader('Allow', ['POST']);
    res.status(405).end(`Method ${req.method} Not Allowed`);
  }
}
```

Step 5.3: Implement Audit Trails and Version Control

Every significant action (data submission, data edit, report generation, role change) should be logged to a dedicated audit_logs collection in Firestore. This provides a complete, immutable history.

- **Conceptual Audit Log Entry:**

```
// Example whenever an action occurs
await addDoc(collection(db, 'audit_logs'), {
  userId: user.uid,
  action: 'DATA_SUBMITTED',
  details: {
    dataType: 'energyConsumption',
    value: energyConsumption,
    docId: newDocRef.id,
  },
  timestamp: new Date(),
});
```

Expected Outcome (Phase 5)

The platform will be able to generate comprehensive ESG report drafts, leveraging AI for narrative content. All data changes and report generations will be meticulously logged, providing a complete audit trail for compliance and verification.

Phase 6: Continuous Improvement & Monitoring

The final phase focuses on making the platform dynamic and responsive, enabling continuous monitoring of ESG performance and automatic alerts for deviations or emerging risks.

Objective

To establish a feedback loop for continuous ESG performance improvement, real-time monitoring, and proactive alerting.

Key Features

- Real-time monitoring of key ESG KPIs.
- Automated alerts for threshold breaches or significant anomalies.

- Feedback mechanism for AI model improvements.
- Integration with external monitoring tools (conceptual).
- Dashboard for overall ESG performance health.

Technology Focus

- **Firebase Firestore:** Real-time listeners for data changes.
- **Next.js:** For displaying real-time updates and alerts.
- **Google Cloud Functions:** For backend triggers (e.g., sending alerts when a Firestore document changes).
- **Gemini API/Genkit:** For continuous re-evaluation and refinement of insights.

Step-by-Step Implementation (Conceptual)

Step 6.1: Real-time Monitoring and Alerts

Use Firestore's real-time capabilities (onSnapshot) in your Next.js frontend to update dashboards instantly. For server-side alerts (e.g., email/SMS notifications), use Google Cloud Functions triggered by Firestore changes.

- **Conceptual Cloud Function (Node.js):**

```
// functions/index.js (Firebase Cloud Function)
const functions = require('firebase-functions');
const admin = require('firebase-admin');
admin.initializeApp();

exports.monitorEsgData = functions.firestore
  .document('esg_data/{docId}')
  .onUpdate(async (change, context) => {
    const newValue = change.after.data();
    const previousValue = change.before.data();

    const energyThreshold = 2000; // Example threshold
    if (newValue.energyConsumption > energyThreshold &&
previousValue.energyConsumption <= energyThreshold) {
      // Send an alert
      console.log(`High energy consumption alert for user ${newValue.userId}:
${newValue.energyConsumption} kWh`);
      // You would integrate with an email/SMS service here
      // e.g., await sendEmailAlert(newValue.userId, "High Energy Consumption", `Energy
consumption exceeded ${energyThreshold} kWh.`);
    }

    // You can also re-run AI analysis on updates if needed
    // const aiAnalysis = await callGeminiAPIForReanalysis(newValue);
    // await change.after.ref.update({ aiAnalysis: aiAnalysis });
```

```
});
```

Step 6.2: AI Model Feedback Loop

Implement a mechanism for users (e.g., data managers, auditors) to provide feedback on AI-generated insights or extracted data. This feedback can be stored and used to retrain or fine-tune your Gemini models or custom ML models over time, improving accuracy.

- **Conceptual Feedback Form:** A simple form on the dashboard where users can rate AI analysis or suggest corrections. Store this in a feedback collection in Firestore.

Expected Outcome (Phase 6)

The platform will actively monitor ESG performance, alert stakeholders to critical changes, and continuously improve its AI capabilities through a feedback loop, ensuring the platform remains accurate, relevant, and proactive.

Conclusion: Your End-to-End AI-Powered ESG Platform

By following these phased steps, you will build a robust, AI-powered ESG management platform that covers:

- **Data Gathering:** Manual entry, external API integrations, and automated extraction from unstructured documents using AI.
- **Data Cleaning and Usage:** AI-powered anomaly detection, standardization, and structured storage.
- **Integrations:** Firebase, Google Cloud Storage, Gemini API, and Genkit.
- **AI Analysis, Insights, and Improvements:** Predictive analytics, risk forecasting, materiality assessment, and operational optimization.
- **Dashboard:** Intuitive, customizable visualizations for different stakeholders.
- **User Management:** Secure authentication with Role-Based Access Control (RBAC).
- **Automatic Compliance Report Maker:** Automated drafting, framework alignment, and XBRL integration.
- **Auditability:** Comprehensive audit trails and version control.
- **Continuous Monitoring:** Real-time alerts and a feedback loop for ongoing AI model refinement.

This end-to-end solution provides a powerful tool for organizations to manage their ESG performance strategically, moving beyond mere compliance to drive real business value and sustainability leadership.