**Deployment Plan for Soil Farming Agent**

**1. Cloud Hosting Selection**

* Choose a cloud platform: **Firebase Hosting** (for frontend) and **Firebase Firestore** (for database).
* Alternative: **AWS S3 + AWS Lambda + DynamoDB** for a scalable setup.

**2. Steps to Deploy on Firebase**

1. **Install Firebase CLI**
2. npm install -g firebase-tools
3. **Login to Firebase**
4. firebase login
5. **Initialize Firebase** (Run in project root directory)
6. firebase init
   * Select **Hosting** and **Firestore**.
   * Choose public as the directory for deployment.
   * Configure as a single-page app (SPA) if necessary.
7. **Deploy the Project**
8. firebase deploy

**3. Database Setup (Firestore Rules)**

rules\_version = '2';

service cloud.firestore {

match /databases/{database}/documents {

match /users/{userId} {

allow read, write: if request.auth != null;

}

match /admin/{adminId} {

allow read, write: if request.auth.token.admin == true;

}

}

}

**4. Domain & SSL Configuration**

* Use **Firebase Custom Domains** to attach a custom domain.
* Firebase provides **free SSL certificates** for secure HTTPS.

**5. Monitoring & Logging**

* Use **Firebase Analytics** for user tracking.
* Enable **Firebase Performance Monitoring**.
* Set up **Cloud Logging** for errors.

**6. CI/CD Automation**

* Use **GitHub Actions** or **Firebase Hosting GitHub Action** for auto-deployment.

**Low-Level Design (LLD) for Soil Farming Agent**

**1. Database Schema (Firestore Structure)**

**Users Collection**

users: {

userId: {

"name": "John Doe",

"email": "john@example.com",

"password": "hashed\_password",

"role": "user",

"createdAt": "timestamp"

}

}

**Admins Collection**

admins: {

adminId: {

"name": "Admin Name",

"email": "admin@example.com",

"password": "hashed\_password",

"role": "admin",

"createdAt": "timestamp"

}

}

**Soil Data Collection**

soilData: {

soilId: {

"userId": "ref\_to\_user",

"soilType": "Loamy",

"pH": 6.5,

"moisture": 30,

"recommendations": "Use organic fertilizers",

"createdAt": "timestamp"

}

}

**2. API Design (Endpoints)**

**User Authentication**

* **POST /register** → User Signup
* **POST /login** → User Login

**Admin Authentication**

* **POST /admin/login** → Admin Login

**Soil Data Management**

* **POST /soilData** → Add Soil Data
* **GET /soilData/{userId}** → Fetch Soil Data for a User

**3. User Dashboard Functionalities**

* **View Soil Reports**
* **View Distributor Information**
* **Logout**

**4. Admin Dashboard Functionalities**

* **Manage Soil Data**
* **Logout**

**System Architecture for Soil Farming Agent**

**1. System Flowchart**

**Flow:**

1. User/Admin visits the login page.
2. Authentication via Firebase Authentication.
3. Users can view/add soil data.
4. Admins can manage users and soil data.
5. Data stored/retrieved from Realtime Database.

**2. Wireframe Overview**

* **Home Page:** Registration and login buttons.
* **Registration / Login Page**: Fields for email & password, login button.
* **User Dashboard**: Displays soil data, recommendations, and profile options.
* **Admin Dashboard**: View user list, manage soil data.

**3. System Components**

**Frontend (Client-Side)**

* HTML, CSS, JavaScript (Vanilla JS)
* Firebase Authentication (for user/admin login)

**Backend (Server-Side)**

* Firebase Realtime (Database)
* Firebase Functions (if needed for advanced logic)

**Security Measures**

* Role-based authentication
* Firestore security rules
* Data validation before storing

This document provides an overview of system design, including flowchart, wireframes, and component architecture.

**Solution Optimization Report for Soil Farming Agent**

**1. Code Optimization**

**Before Optimization:**

function fetchData() {

fetch("https://firestore.googleapis.com/v1/projects/project-id/databases/(default)/documents/soilData")

.then(response => response.json())

.then(data => console.log(data))

.catch(error => console.error(error));

}

**After Optimization:**

async function fetchData() {

try {

let response = await fetch("https://firestore.googleapis.com/v1/projects/project-id/databases/(default)/documents/soilData");

let data = await response.json();

console.log(data);

} catch (error) {

console.error(error);

}

}

**Improvement:** Switched to async/await for better readability and error handling.

**2. Architecture Optimization**

* **Before:** Direct database access from frontend → security risk.
* **After:** Implemented **Firebase Functions** to handle sensitive operations securely.

**3. Performance Enhancements**

* **Reduced API calls** by implementing caching in the frontend.
* **Enabled Firestore indexing** for faster queries.
* **Minimized bundle size** by removing unused JS dependencies.

**4. Security Enhancements**

* **Firestore Security Rules:** Applied strict rules to prevent unauthorized access.
* **Role-Based Access Control (RBAC):** Users can only access their own data.
* **Form Validation:** Implemented input validation to prevent injection attacks.

**Test Cases for Soil Farming Agent**

**1. User Authentication Tests**

**1.1 User Login**

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Description** | **Expected Result** | **Status** |
| 1 | Login with valid credentials | User successfully logs in | Successful |
| 2 | Login with incorrect password | incorrect password displayed | Successful |
| 3 | Login with non-registered email | Error message displayed | Successful |

**1.2 Admin Login**

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Description** | **Expected Result** | **Status** |
| 4 | Admin logs in with valid credentials | Admin dashboard loads | Successful |
| 5 | Admin logs in with incorrect credentials | Error message displayed | Successful |

**2. Soil Data Management Tests**

**2.1 Adding Soil and Distributor Data**

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Description** | **Expected Result** | **Status** |
| 6 | Admin adds valid data | Data successfully stored | Successful |
| 7 | Admin adds invalid data | Error message displayed | Pending |

**2.2 Fetching Soil Data**

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Description** | **Expected Result** | **Status** |
| 8 | User fetches data | Data displayed correctly | Successful |

**3. Security Tests**

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Description** | **Expected Result** | **Status** |
| 9 | Attempt unauthorized API access | Access denied | Successful |

**Detailed Project Report: Soil Farming Agent**

**1. Introduction**

The Soil Farming Agent is a web-based system designed to help farmers analyse soil data and receive recommendations for optimal crop production. The system supports both user and admin roles for managing and monitoring soil data effectively.

**2. Objectives**

* Provide an easy-to-use interface for soil data management.
* Enable users to analyse soil properties like pH and moisture.
* Ensure data security with Firebase Authentication and Firestore rules.
* Offer admin functionalities for managing user data.

**3. System Overview**

* **Frontend:** HTML, CSS, JavaScript
* **Backend:** Firebase Firestore
* **Authentication:** Firebase Authentication
* **Hosting:** Firebase Hosting

**4. System Features**

**User Features:**

* Register/Login using Firebase Authentication.
* Input and view soil data.
* Get automated recommendations for soil improvement.

**Admin Features:**

* Manage user accounts.
* View and analyse overall soil data.

**5. System Architecture**

* **Client-side:** Handles UI interactions and API calls.
* **Database (Realtime):** Stores users and soil data.
* **Security:** Role-based authentication ensures data protection.

**6. Deployment Plan**

* Firebase Hosting for web application.
* Realtime Database for scalable data management.

**7. Testing & Optimization**

* Implemented structured test cases for login, soil data management, and security.
* Optimized API calls to improve performance.
* Ensured role-based security for data access.

**8. Conclusion**

The Soil Farming Agent system provides a comprehensive platform for soil data analysis, benefiting farmers with insightful recommendations while maintaining a secure and scalable architecture.

**9. Reference:**

Websites:

* <https://chatgpt.com/>
* <https://www.google.com/>
* <https://www.youtube.com/>
* <https://console.firebase.google.com/>
* <https://github.com/>
* <https://www.resizepixel.com/>
* <https://fontawesome.com/>

Tools:

* Visual Studio Code
* Live Server
* Google Chrome Browser