
🌱 Farmer Soil Testing App – Salesforce Code Documentation

📘 Overview

The **Farmer Soil Testing App** is a Salesforce-based application aimed at helping farmers manage and monitor the health of their soil. It enables them to:

- Register their land and crops.
- Submit soil sample data.
- Receive reports with soil metrics (e.g., pH, Nitrogen).
- Get automated recommendations on fertilizers and crop types.

This document explains the structure and code used in building this app, including:

- Custom Object schema
- Apex server-side logic
- Lightning Web Components (LWC) for the user interface
- Integration considerations

🧱 1. Salesforce Custom Objects

Salesforce uses **Custom Objects** to store structured data. We define three main custom objects for this app.

1.1 Farmer__c

Represents an individual farmer.

Fields:

- Name: Text – Full name of the farmer
- Contact__c: Lookup(Contact) – Link to the Salesforce Contact object
- Location__c: Text – Region or GPS location
- Farm_Size__c: Number – Total area of the farm (in acres/hectares)

1.2 Soil_Sample__c

Stores individual soil test samples.

Fields:

- Farmer__c: Lookup(Farmer__c)
- Sample_Date__c: Date

- Status__c: Picklist (Pending, Analyzing, Completed)
- pH_Level__c, Nitrogen__c, Phosphorus__c, Potassium__c: Decimal fields for soil chemistry

1.3 Recommendation__c

Holds expert or system-generated suggestions based on soil quality.

Fields:

- Soil_Sample__c: Lookup to the soil sample
- Crop_Type__c: Picklist (Wheat, Rice, Vegetables, etc.)
- Recommendation_Text__c: Long text area containing advice

⚙️ **2. Apex Classes (Business Logic)**

Apex is Salesforce's proprietary programming language used for backend logic. Here's the main controller.

SoilTestController.cls

```
public with sharing class SoilTestController {

    // Get all soil samples related to a farmer
    @AuraEnabled(cacheable=true)
    public static List<Soil_Sample__c> getFarmerSamples(Id farmerId) {
        return [
            SELECT Id, Sample_Date__c, Status__c, pH_Level__c, Nitrogen__c, Phosphorus__c,
Potassium__c
            FROM Soil_Sample__c
            WHERE Farmer__c = :farmerId
            ORDER BY Sample_Date__c DESC
        ];
    }

    // Insert a new soil sample record
    @AuraEnabled
    public static void submitSoilSample(Soil_Sample__c sample) {
        sample.Status__c = 'Pending';
        insert sample;
```

```
    }

    // Retrieve recommendation for a specific sample
    @AuraEnabled(cacheable=true)
    public static Recommendation__c getRecommendation(Id sampleId) {
        return [
            SELECT Id, Recommendation_Text__c, Crop_Type__c
            FROM Recommendation__c
            WHERE Soil_Sample__c = :sampleId
            LIMIT 1
        ];
    }
}
```

Explanation:

- @AuraEnabled: Exposes methods to Lightning components.
- getFarmerSamples(): Queries records for a farmer's samples.
- submitSoilSample(): Creates a new soil sample with default status.
- getRecommendation(): Fetches crop advice tied to the sample.

---

3. Lightning Web Components (LWC)

LWCs are Salesforce's modern JavaScript-based UI components.

3.1 Metadata: farmerDashboard.js-meta.xml

```
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>59.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__AppPage</target>
        <target>lightning__RecordPage</target>
    </targets>
</LightningComponentBundle>
```

This configuration allows the component to be placed on farmer record and app pages.

3.2 UI Template: farmerDashboard.html

```
<template>
  <lightning-card title="Soil Samples" icon-name="custom:custom63">
    <template if:true={samples}>
      <template for:each={samples} for:item="sample">
        <div key={sample.Id} class="slds-box slds-box_xx-small">
          <p><strong>Date:</strong> {sample.Sample_Date__c}</p>
          <p><strong>Status:</strong> {sample.Status__c}</p>
          <p><strong>pH:</strong> {sample.pH_Level__c},
            <strong>Nitrogen:</strong> {sample.Nitrogen__c}</p>
        </div>
      </template>
    </template>
    <template if:false={samples}>
      <p>Loading samples...</p>
    </template>
  </lightning-card>
</template>
```

3.3 JS Controller: farmerDashboard.js

```
import { LightningElement, api, wire } from 'lwc';
import getFarmerSamples from '@salesforce/apex/SoilTestController.getFarmerSamples';

export default class FarmerDashboard extends LightningElement {
  @api recordId; // Farmer record ID
  samples;

  @wire(getFarmerSamples, { farmerId: '$recordId' })
  wiredSamples({ error, data }) {
    if (data) {
      this.samples = data;
```

```
        } else if (error) {
            console.error('Error fetching samples:', error);
        }
    }
}
```

🧠 4. Application Flow

Here's how the app works:

1. **Farmer logs in** to view their dashboard.
2. **Farmer enters a new soil sample** (optionally via mobile app).
3. **Apex logic stores the sample**, sets the status to Pending.
4. A lab or automated script updates the **sample values**.
5. **Recommendations are generated** based on nutrient levels.
6. **Farmer views the suggestion** using the LWC dashboard.

🔌 5. Integration Options

This app can integrate with:

- **Salesforce Mobile App** – for on-field access
- **External Labs** – via REST API to import test results
- **Einstein Prediction Builder** – for smarter crop advice
- **Weather APIs** – to enhance recommendations

🚀 6. Future Enhancements

Feature	Benefit
Geolocation	Tag samples to specific farm areas
Image Upload	Attach soil sample or crop condition images
Push Notifications	Alert farmers when results are available
Offline Support	Allow sample submission in low-connectivity zones
Multilingual UI	Broaden reach to rural farmers

📝 **Conclusion**

The **Farmer Soil Testing App** demonstrates how Salesforce can be tailored to solve real-world agricultural challenges. Using **Custom Objects** to manage data, **Apex** for server-side processing, and **LWC** for UI, it delivers an efficient and scalable solution for soil health management.

This system is fully extensible and serves as a foundation for future innovations like AI-driven recommendations, IoT soil sensors, and integration with government agriculture systems.

Additional tools codes:

Emailing to the user:

🛠️ **1. Using Salesforce Flow (No Code)**

Use Case:

Send automated emails when a record is created, updated, or a condition is met.

Steps:

1. Go to **Setup → Flows**.
2. Click **New Flow** and select **Record-Triggered Flow**.
3. Choose the object (e.g., Contact, Case, or custom object).
4. Set the trigger (e.g., when record is created).
5. Add an **Action → Send Email**.
6. Configure:
    - **Email Template** (optional)
    - **Recipient**: Record.Email or user email
    - **Subject** and **Body**
7. Save and Activate the flow.

✅ **Best for:** Admins, non-developers, and basic automation.

💻 **2. Using Apex Code (Developer Option)**

Salesforce provides the Messaging class to send emails.

Example Apex Code:

apex

CopyEdit

```
public class SendEmailController {

    public static void sendEmailToUser(Id userId) {
        User user = [SELECT Email FROM User WHERE Id = :userId LIMIT 1];

        Messaging.SingleEmailMessage email = new Messaging.SingleEmailMessage();
        email.setToAddresses(new String[] {user.Email});
        email.setSubject('Important Update from Soil Testing App');
        email.setPlainTextBody('Hello! Your soil test results are ready. Please log in to view them.');

        Messaging.sendEmail(new Messaging.SingleEmailMessage[] { email });
    }
}
```

✅ **Best for:** Custom triggers, batch jobs, or LWC integrations.

📨 **3. Using Email Alerts (Workflow Rules / Process Builder)**

Use Case:

Send an email automatically when a field is updated.

Steps:

1. Go to **Setup → Email Alerts** and create a new one.
2. Create or use an existing **Email Template**.
3. Use **Workflow Rule** or **Process Builder** to trigger the alert.
4. Set criteria (e.g., Status = Completed).
5. Add the **Email Alert** as an action.

✅ **Best for:** Simple automated notifications.

🧠 **4. Using Email Templates**

You can design professional emails with dynamic fields using:

- **Text**
- **HTML (with letterhead)**
- **Visualforce Email Templates** (for advanced use)

Create One:

1. Go to **Setup → Email Templates**.
2. Click **New Template**.
3. Choose type and design the body using merge fields like:

html

CopyEdit

Hello {!Contact.FirstName},

Your soil test results are now available.

Then, use this template in flows, alerts, or Apex.

🔒 **Notes & Limits**

- Salesforce has **daily email limits** (e.g., 5,000 per org for mass email).
- Use **Organization-Wide Email Addresses** to customize the sender.
- Verify that users have **valid email addresses** and **email deliverability** is set to **All** in Setup.

🧪 **Testing Emails in Sandbox**

- Go to **Setup → Deliverability** and set to **All Emails**.
- Use test users with your own email for safety.
- Use System.debug() to log email details in Apex.